WO0150612

Publication Title:

SYSTEMS AND METHODS FOR MULTIPLE-FILE DATA COMPRESSION

Abstract:

Abstract of WO 0150612

(A1) Translate this text The systems and methods relate to the compression of multiple files into a single file called an archive. Before appending the multiple files as one file to be compressed, the systems and methods arrange the order of files to increase the potential of redundancy among neighboring files, thus providing potential improvement in compression ratio and compression speed. In using a dictionary method to compress the multiple files appended as one file, a large dictionary is used in one embodiment to take advantage of potential between-file redundancies. In another embodiment, the redundancy characteristics of the multiple files are examined to dynamically determine the dictionary size. After the dictionary compression method produces an intermediary output data file, the intermediary output data may be separated into multiple sections, and a compression method that is potentially suitable for the data characteristics of each section may be applied. The compressed results of each section are then combined to produce the final output.

---

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
12 July 2001 (12.07.2001)

PCT

(10) International Publication Number
WO 01/50612 A1

(51) International Patent Classification[7]: H03M 7/30, 7/40

(21) International Application Number: PCT/US01/00424

(22) International Filing Date: 5 January 2001 (05.01.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/174,518 5 January 2000 (05.01.2000) US

(71) Applicant: REALNETWORKS, INC. [US/US]; 2601 Elliott Avenue, Seattle, WA 98121 (US).

(72) Inventor: BELU, Sabin; 1705 Belmont Avenue #302, Seattle, WA 98122 (US).

(74) Agent: ALTMAN, Daniel, E.; Knobbe, Martens, Olson & Bear, LLP, 620 Newport Center Drive, 16th Floor, Newport Beach, CA 92660 (US).

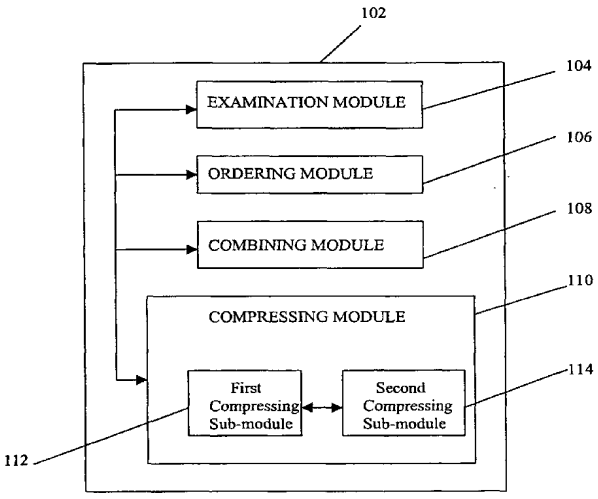
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, CZ (utility model), DE, DE (utility model), DK, DK (utility model), DM, DZ, EE, EE (utility model), ES, FI, FI (utility model), GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SK (utility model), SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *With international search report.*
— *Before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments.*

[Continued on next page]

(54) Title: SYSTEMS AND METHODS FOR MULTIPLE-FILE DATA COMPRESSION



(57) Abstract: The systems and methods relate to the compression of multiple files into a single file called an archive. Before appending the multiple files as one file to be compressed, the systems and methods arrange the order of files to increase the potential of redundancy among neighboring files, thus providing potential improvement in compression ratio and compression speed. In using a dictionary method to compress the multiple files appended as one file, a large dictionary is used in one embodiment to take advantage of potential between-file redundancies. In another embodiment, the redundancy characteristics of the multiple files are examined to dynamically determine the dictionary size. After the dictionary compression method produces an intermediary output data file, the intermediary output data may be separated into multiple sections, and a compression method that is potentially suitable for the data characteristics of each section may be applied. The compressed results of each section are then combined to produce the final output.

WO 01/50612 A1

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

SYSTEMS AND METHODS FOR MULTIPLE-FILE DATA COMPRESSION

Field of the Invention

This invention relates to the field of data compression. In particular, it relates to systems and methods of organizing multiple files to be compressed into one archive file, and systems and methods of compressing the multiple files as one unified file.

Background

A number of compression methods have been utilized to reduce the size of an input data file by encoding symbols or strings of symbols as numeric codes or combinations of numeric codes and symbols. Such encoding reduces repetitions of symbols in the input data file. Major categories of compression methods include dictionary methods and statistical methods. Dictionary methods encode strings of symbols as tokens using a dictionary. A token indicates the location of the string in the dictionary. Dictionary methods include LZ77, LZ78, LZSS, and LZW. Statistical methods use codes of variable lengths to represent symbols or groups of symbols, with the shorter codes representing symbols or groups of symbols that appear or are likely to appear more often. Statistical methods include the Huffman method and the dynamic Huffman method.

The LZ77 method reads part of the input data into a window. The window is divided into a search buffer (i.e., a dictionary) on the left and a look-ahead buffer on the right. The search buffer is the current dictionary that includes symbols that have been read and encoded. The look-ahead buffer includes data yet to be encoded. The encoder scans the search buffer from right to left looking for a match to the longest stream of symbols in the look-ahead buffer. When the longest match is found, the matched stream of symbols plus the next symbol in the look-ahead buffer are encoded as a token containing three parts, the position (i.e., the distance from the end of the search buffer), the length of the longest match, and the next symbol. For a more detailed description of the LZ77 method please refer to pages 154-157 of Data Compression the Complete Reference by David Salomon, Second Edition, 2000. The LZ78 method is a variation of LZ77. For a more detailed description of the LZ78 method please refer to pages 164-168 of Data Compression the Complete Reference by David Salomon, Second Edition, 2000.

The LZSS method is a variation of LZ77. Unlike the LZ77 tokens, LZSS tokens use two fields instead of three. An LZSS token includes a position and a length. If no match is found, the uncompressed code of the next symbol is produced as output, with a flag bit to indicate it is uncompressed. The LZSS method also holds the look-ahead buffer in a circular queue and the search buffer in a binary search tree. For a more detailed description of the LZSS method please refer to pages 158-161 of Data Compression the Complete Reference by David Salomon, Second Edition, 2000.

The LZW method is a variation of LZ78. For a detailed description of the LZW method please refer to U.S. Patent No. 4,558,302 issued on December 10, 1985.

The Huffman method, also called the static Huffman method, builds a list of all of the symbols in descending order according to their probabilities of appearance. The method builds a tree from the bottom up with each leaf representing a symbol. The two symbols with the smallest probabilities of appearance are added to the top of the tree and deleted from the list. An auxiliary symbol is created to represent these two symbols. This process is repeated until the list is reduced to one auxiliary symbol. The method then assigns codes of variable length to the symbols on the tree. One variation of the Huffman method is the dynamic, or adaptive, Huffman method. This method assumes that the probabilities of appearance are not known prior to reading the input data. The compression process starts with an empty tree and modifies the tree as symbols are being read and compressed. The decompression process works in synchronization with the compression process. For a more detailed description of the Huffman method and the dynamic Huffman method please refer to pages 62-82 of Data Compression the Complete Reference by David Salomon, Second Edition, 2000.

Context based methods are another type of compression methods. Context based methods use one or more preceding symbols to predict (i.e., to assign the probability of) the appearance of the next symbol. For example, in English, the letter "q" is almost always followed by the letter "u". When letter "q" appears, a context based method would assign a high probability of appearance to the letter "u". One example of a context based method is the Markov model which may be classified by the number of proceeding symbols it uses to predict the next symbol. An Order-N Markov model uses the N preceding symbols to predict the next symbol. For a more detailed description of the Markov model, please refer to page 126 and pages 726 - 735 of Data Compression the Complete Reference by David Salomon, Second Edition 2000.

Compression methods may be used by programs to compress multiple files to archives. For example, the ARC program compresses multiple files and combines them into one file called an archive. PKZIP is a variation of ARC. ZIP is an open standard created by the maker of PKZIP for compressing files to archives. For details of ARC and PKZIP please refer to pages 206-211 of Data Compression the Complete Reference by David Salomon, Second Edition, 2000. Other compression/archiving programs include ARJ by Robert K. Jung, LHArc by Haruyasu Yoshizaki, and LHZ by Haruhiko Okumura and Haruyasu Yoshizaki.

Summary of the Invention

The systems and methods relate to the compression of multiple files into a single file called an archive. The systems and methods examine the multiple files to determine their data characteristics. The systems and methods then arrange the order of the multiple files according to their data characteristics to increase the potential of data redundancy among neighboring files. The increased potential of redundancy provides potential improvement in compression ratio and compression speed. The ordered multiple files are then combined as one unified file and compressed.

One embodiment uses a dictionary method to compress the unified file. In addition, a large dictionary is used in one embodiment to take advantage of potential between-file redundancies. In another embodiment, the redundancy characteristics of the multiple files are examined to dynamically determine the dictionary size. After the dictionary

compression method produces an intermediary output data file, the intermediary output data may be separated into multiple sections, such that for each section a compression method that is potentially suitable for the data characteristics of that section is applied. The compressed result of each section are then combined to produce the final output.

One embodiment of the present invention is a method of compressing a plurality of files. The method comprises examining said plurality of files to determine data characteristics that correspond to said plurality of files and determining ranking orders for said plurality of files according to said data characteristics. In addition, the method comprises combining said plurality of files into a unified file at least according to said ranking orders and compressing said unified file.

An additional embodiment of the present invention is a system for compressing a plurality of files. The system comprises an examination module configured to examine said plurality of files to determine data characteristics that correspond to said plurality of files and an ordering module configured to determine ranking orders for said plurality of files. In addition, the system comprises a combining module configured to combine said plurality of files as a unified file at least according to the ranking orders of said plurality of files and a compressing module configured to compress said unified file using a first compression method.

A further embodiment of the present invention is a system for compressing a plurality of files. The system comprises means for examining said plurality of files to determine data characteristics that correspond to said plurality of files and means for determining ranking orders for said plurality of files. In addition, the system comprises means for combining said plurality of files as a unified file at least according to the ranking orders of said plurality of files and means for compressing said unified file using a first compression method.

For purposes of summarizing the invention, certain aspects, advantages, and novel features of the invention are described herein. It is to be understood that not necessarily all such advantages may be achieved in accordance with any particular embodiment of the invention. Those skilled in the art will recognize that the invention may be embodied or carried out in a manner that achieves one advantage or a group of advantages as taught herein without necessarily achieving other advantages as may be taught or suggested herein.

These and other features will now be described with reference to the drawings summarized below. These drawings and the associated description are provided to illustrate embodiments of the invention, and not to limit the scope of the invention. Throughout the drawings, reference numbers are re-used to indicate correspondence between referenced elements. In addition, the first digit of each reference number indicates the figure in which the element first appears.

Brief Description of the Drawings

FIGURE 1 illustrates a high-level block diagram for one embodiment of a multiple-file compression system.

FIGURE 2 illustrates one embodiment of an example original file list and one embodiment of an example sorted file list.

FIGURE 3 illustrates one embodiment of a unified input data file, one embodiment of an intermediary output file, and one embodiment of a final output file.

FIGURE 4 illustrates a flowchart of one embodiment of a first-stage compression process.

FIGURE 5 illustrates a flowchart of one embodiment of a second-stage compression process.

Detailed Description

I. OVERVIEW

The present invention relates to the compression of multiple files into a single file called an archive. The data characteristics of the multiple files are examined to determine the order in which the files are ranked in a sorted file list. The files are then combined into a single unified file according to the order of the sorted file list and compressed. The ordering of the files increases the potential of repetition among neighboring files and provides potential improvement in compression ratio and compression speed. In one embodiment, the compressed results are compressed a second time.

One embodiment of the systems and methods compresses the ordered files using a dictionary method and utilizing a large dictionary. Such a large dictionary not only has the potential to improve the compression ratio of files with high degree of redundancy within a file, but also has the potential to improve the compression ratio of neighboring files that are somewhat redundant with each other. Because similar files have been placed next to each other in the sorted file list, a large dictionary is likely to take advantage of between-file redundancy to improve the compression ratio of the files. Another embodiment examines the redundancy characteristics of the multiple files to dynamically determine the dictionary size.

In further compressing the intermediary output data already compressed by a dictionary method, one embodiment separates the intermediary output data into multiple sections based on the characteristics of the intermediary output data. The data within each section may have redundancy. Each section of the intermediary output data is compressed using a method that is potentially suitable for compressing data with such characteristics. The compressed results are then combined with header information to form the final output data.

II. MULTIPLE-FILE COMPRESSION SYSTEM

FIGURE 1 illustrates a high-level block diagram of a multiple-file compression system 102. The system 102 includes an examination module 104, an ordering module 106, a combining module 108 and a compressing module 110. One embodiment of the compressing module 110 also includes a first compressing sub-module 112 and a second compressing sub-module 114.

As used herein, the word module, whether in upper or lower case letters, refers to logic embodied in hardware or firmware, or to a collection of software instructions, possibly having entry and exit points, written in a programming language, such as, for example, C++. A software module may be compiled and linked into an executable program, installed in a dynamic link library, or may be written in an interpretive language such as BASIC. It will be appreciated that software modules may be callable from other modules or from themselves, and/or may be invoked in response to detected events or interrupts. Software instructions may be embedded in firmware, such as an EPROM. It will be further appreciated that hardware modules may be comprised of connected logic units, such as gates and flip-flops, and/or may be

comprised of programmable units, such as programmable gate arrays or processors. The modules described herein are preferably implemented as software modules, but may be represented in hardware or firmware.

It should be obvious to those skilled in the art that the modules described in the application may be integrated into fewer modules. One module may also be separated into multiple modules. It should also be obvious to those skilled in the art that the described modules can be implemented as hardware, software, firmware or any combinations of. The described modules may reside at different locations connected through a wired or wireless network.

A. Examination Module

The examination module 104 prompts a user to identify an original file list 202. An example of the original file list 202 is illustrated in FIGURE 2 and will be discussed further below. Referring back to FIGURE 1, the examination module 104 examines the files in the original file list 202 to determine the data characteristics of each file. The examination may include reading all or part of each file, reading attribute/properties information of each file, reading the size, location, name and/or extension of each file, as well as a combination of the above. In one embodiment, the examination module 104 examines the appearance frequencies of "a-Z" alphabets. In another embodiment, the examination module 104 examines the appearance frequencies of numbers.

B. Ordering Module

The ordering module 106 orders files of the original file list 202 into a sorted file list 202. An example of the sorted file list 204 is illustrated in FIGURE 2 and discussed further below. Referring back to FIGURE 1, files in the sorted file list 204 are ordered according to their data characteristics examined by the examination module 104. Files with similar data characteristics are placed next to each other. The examination module 104 may examine multiple data characteristics of each file, and the ordering module 106 may determine the ordering of the files according to some or all of their multiple data characteristics. For example, files may be sorted by file extension first and then by file name. For another example, files may be sorted by file extension first and then by file size. In one embodiment, the ordering module 106 automatically orders files in the sorted file list. In another embodiment, the ordering module 106 acts as a wizard and provides a user with help information to assist the user in determining the order of the files. Help information may include, for example, the data characteristics of the files.

In addition to using the sorted file list 204, other embodiments may also be used to record the order of files of the original file list 202. For example, each file of the original file list 202 may be assigned an order number indicating the order in which it would be combined by the combining module 108.

C. Combining Module

The combining module 108 combines the files successively according to the order in the sorted file list 204, to form a unified input data file 302. An example of the unified input data file 302 is illustrated in FIGURE 3 and is discussed further below. Referring back to FIGURE 1, the combining module 108 removes the "End-of-File" (i.e., "EOF") marker from

each file except for the last file in the sorted file list 204. In one embodiment the combining module 108 also creates an archive header portion 304 and a file header portion 306.

D. Compressing Module

The compressing module 110 applies a first compression method to the unified input data file 302 produced by the combining module 108. In one embodiment, the compressing module 110 applies a dictionary method. The compressing module 110 examines the files in the sorted file list 204 and determines a dictionary size that is likely to be optimal with the redundancy characteristics of the files in the sorted file list 204. In one embodiment, the determination of the dictionary size is made by the examination module 104. The determination of the dictionary size may also be made by the ordering module 106 or by the user. In one embodiment, the compressing module 110 also compresses the file header portion 306 to produce a compressed file header portion 324. The compressing module 110 combines the archive header 304, the compressed file header portion 324 and the compressed result of the unified input data file 302 to produce the final output file 322.

In one embodiment, the compressing module 110 applies a second compression method to the intermediary output file 312 produced by the first compression method. In one embodiment, the compressing module 110 separates the intermediary output file 312 into multiple sections and compresses the data in one or more of the sections. The compressing module 110 then combines the compressed or uncompressed results of all the sections to form the final compressed data portion 326. The compressing module 110 combines the final compressed data portion 326, the archive header portion 304, and the file header portion 306 or the compressed file header portion 324 to form the final output file 322. In one embodiment, a recovery information portion 328 is also appended to the final output file 322. An example of the final output file 322 is illustrated in FIGURE 3 and is further discussed below.

One embodiment of the compressing module 110 includes a first compressing sub-module 112 and a second compressing sub-module 114. The first compressing sub-module 112 is configured to apply a first compression method to the unified input data file 302. The second compressing sub-module 114 is configured to apply a second compression method to the intermediary output file 312 produced by the first compression method.

In one embodiment, the compressing module 110 combines the final output file 322 with a ZIP archive header 334, a ZIP central directory header 336, and a ZIP end of central directory header 338 to form a ZIP-recognized output file 332. The ZIP-recognized output file 332 may be recognized by any utility program that supports the ZIP file format. Details of the ZIP headers are described below in the section titled "III. FILE FORMATS."

III. FILE FORMATS

FIGURE 2 illustrates an example of an original file list 202 and an example of a sorted file list 204. In this example, the original file list 202 of six files is sorted to produce the sorted file list 204 of the same six files in the same or a different order.

FIGURE 3 illustrates an embodiment of a unified input data file 302, an embodiment of an intermediary output file 312, and an embodiment of a final output file 322. FIGURE 3 also illustrates an embodiment of an archive header portion 304, an embodiment of a file header portion 306, and an embodiment of a ZIP-recognized output file 332. The exemplary unified input data file 302 is formed by combining all the files in the sorted file list 304. The exemplary intermediary output file 312 includes the compressed result of the unified input data file 302, produced by the first-stage compression process described below.

The exemplary final output file 322 includes an archive header portion 304 that stores information about the archive, a compressed file header portion 324, a final compressed data portion 326, and a recovery information portion 328. The compressed file header portion 324 includes the compressed result of the file header portion 306. The file header portion 306 includes header information about the files in the sorted file list 104, such as the name, location, size and the date and time of creation or modification of each file, and so forth. The final compressed data portion 326 includes the final compressed result of the intermediary output file 312. The recovery information portion 328 includes recovery information about the final compressed data portion 326. The recovery information may include the CRC (Cyclical Redundancy Check or Cyclical Redundancy Code) of the final compressed data portion 326, and in one embodiment, the recovery information portion 328 may be compressed.

While FIGURE 3 illustrates embodiments of a unified input data file 302, an intermediary output file 312, and a final output file 322, it is recognized that other embodiments may be used. For example, one embodiment of the final output file 322 may not include a recovery information portion 328.

In one embodiment, a ZIP archive header 334, a ZIP central directory header 336 and a ZIP end of central directory header 338 are combined with the final output file 322 to form a ZIP-recognized output file 332. The formats of the ZIP archive header 334, the ZIP central directory header 336 and the ZIP end of central directory header 338 are maintained as open standards by PKWARE, Inc. located in Brown Deer, Wis. (web site www.pkware.com) The ZIP-recognized output file 332 conforms to the current ZIP file format and can therefore be recognized by any utility program that supports the current ZIP file format.

IV. MULTIPLE FILE COMPRESSION PROCESSES

In one embodiment, the multiple file compression process includes a first-stage compression process and a second-stage compression process.

A. First-Stage Compression

1. Prompt For Original File List

FIGURE 4 illustrates one embodiment of a first-stage compression process. Beginning at start state block 400, the first-stage compression process proceeds to block 402 and prompts a user for an original file list 202, which lists the files to be compressed.

2. Create Sorted File List

At block 404, the first-stage compression process converts the original file list 202 into a sorted file list 204 to increase the potential of redundancy among neighboring files in the sorted file list 204. In one embodiment, the original file list 202 is sorted by file extension. For example, files that share the ".doc" extension (i.e., word processing document files) are placed together and files that share the ".cpp" extension (i.e., C++ source code files) are placed together. Because files with the same extension are likely to share more redundancy than files with different extensions, conversion from the original file list 202 into a sorted file list 204 is likely to improve the compression ratio and speed for the first-stage compression. In one embodiment, the original file list 202 is sorted by a combination of file extension and file name. The sorting by file name may further improve the redundancy among neighboring files, such as, for example if some neighboring files are back-up versions of the same master file. In another embodiment, the original file list 202 is sorted by a combination of file extension and file size. For example, files are first sorted by extension, and files of the same extension are then ordered by file size.

In other embodiments, files with the same file type are placed close together in the sorted file list 204. For example, files with the ".doc" extension are placed next to files with the ".txt" extension because they are determined to have the same "text document" file type. Files with the ".exe" extension are placed next to files with the ".dll" extension because they are determined to have the same "binary program" file type. Each file's file type may be determined by using its file extension to look for the corresponding file type in a file extension-file type matching table. The file extension-file type matching table may be created prior to block 402 to assign a file type for each file extension.

In other embodiments, files of the same file group are placed close together in the sorted file list 204. File groups may include, for example, an alphabet-rich file group, a number-rich file group, an image data file group, an audio data file group, a video data file group, a high-redundancy file group, a low-redundancy file group, and so forth. Ordering files by file types and file groups may improve compression ratio and speed because files with the same file type or file group are likely to have more redundancy than other files. File characteristics such as file type and file group may be determined by reading the file extension of a file, reading all or part of a file, reading attribute (also called properties) information about a file, and so forth. Attribute and/or properties information about a file may be included in a header of the file.

In another embodiment, black and white image files may be placed next to each other in the sorted file list 204 and color image files may also be placed next to each other in the sorted file list 204. The color scheme of each image file

may be determined by reading the color scheme information from the image file header or by reading all or part of an image file.

In additional embodiments, an artificial intelligence program, a heuristic program, and/or a rule-based program may be used to determine the file orders in the sorted file list 204. For example, the first-stage compression process may read a part of each file in the original file list 202 to determine the data characteristics of each file. Files with similar redundancy characteristics are placed close together in the sorted file list 204. Data characteristics may include the appearance frequency of the "a-Z" alphabet, the appearance frequency of numbers, the appearance frequency of binary codes, and so forth.

In some embodiments, the first-stage compression process automatically carries out the ordering process of block 404. In other embodiments, a user arranges the ordering of files using help information provided by the first-stage compression process. The help information may include the file extension, file type, file group and other data characteristics of each file. Combinations of automatic ordering and manual ordering may also be used. For example, the first-stage compression process may carry out a preliminary ordering of the original file list 202 and may enable the user to adjust the preliminary orders to produce the sorted file list 204.

In addition to using the sorted file list 204, other embodiments may also be used to record the order of files of the original file list 202. For example, each file of the original file list 202 may be assigned an order number indicating the order in which it would be combined at block 408.

3. Create Headers

Still referring to FIGURE 4, at block 406, the first-stage compression process creates an archive header portion 304 and a file header portion 306. The archive header portion 304 includes information about the entire archive that is to be created. Such information may include the creation date and time of the archive, the archive's CRC (Cyclical Redundancy Check or Cyclical Redundancy Code), a flag indicating whether the archive is locked, a flag indicating whether the archive is damage protected and so forth. In one embodiment, archive information in the archive header portion 304 may be updated after the compressing of block 410. In another embodiment, archive information in the archive header portion 304 may be updated after the second-stage compression illustrated in FIGURE 5 and discussed in further detail below. The file header portion 306 includes file header information for each file in the sorted file list 204. In one embodiment, the file header information for a file may include the file name, the location of the file, the dates and times that the file was created and/or modified, file size prior to compression, and other attribute information about the file such as whether it is read-only.

4. Combine Files

At block 408 of FIGURE 2, the first-stage compression process appends all but the first file in the sorted file list 204 successively to the first file in the sorted file list 204. During the appending process, the "End-of-File" (i.e., "EOF")

marker, if such a marker exists, is removed from each of the files within the sorted file list 204, except the last file in the sorted file list 204. The data from all the files in the sorted file list 204 forms the unified input data file 302. It is recognized that the first-stage compression process may combine the files using a variety of methods, such as, for example, combining subsets of the set of files in parallel and then merging the combined files together.

5. Compress the Combined File

At block 410 of FIGURE 4, the first-stage compression process applies a compression method to the unified input data file 302. A dictionary method, such as the LZSS method, is used in one embodiment, but other compression methods including other dictionary methods may also be applied. The dictionary size may be pre-determined prior to the start of the first-stage compression process, or dynamically determined after the start of the first-stage compression process. The dictionary size may exceed a dictionary size unit number of fixed size, such as, for example, one kilobyte, one megabyte, ten megabytes, a hundred megabytes, one gigabyte, and so forth. As the dictionary size grows, the compression ratio may improve but dictionary search speed may decrease. This is because a string of symbols is more likely to find a match in a larger dictionary than in a smaller dictionary, but a large dictionary takes longer to search.

In one embodiment, the first-stage compression process enables a user to specify a dictionary size for compression. This option permits a user to choose a dictionary size that is likely to be optimal with respect to the redundancy characteristics of the sorted file list 204. In another embodiment, the first-stage compression process examines the files of the sorted file list 204 and determines a dictionary size that is likely to be optimal with respect to the redundancy characteristics of the sorted file list 204. In yet another embodiment, the first-stage compression process provides a user with help information, such as the redundancy characteristics, and assists a user to set the dictionary size.

To determine the redundancy characteristics of the sorted file list 204, the first-stage compression process examines information of the sorted file list 204, such as the total size of all files, the size of each file, how many files share the same file group, how many files share the same file type, how many files share the same file name, how many files have similar file names, the number of different file groups and different file types, and so forth. For example, if several files have the same file extension, very similar file names, and very similar file sizes, then these several files may be back-up versions of the same master file. Therefore, the dictionary size may be set at sufficiently more than the file size of each of the several files to take advantage of the high degree of redundancy.

In one example, the first file is read into the dictionary, (because the dictionary size is large enough to allow the entire first file to be read into the dictionary), the rest of the several files may each be encoded as one token with a length close to the size of the first file, plus a few other tokens and/or uncompressed symbols. In another example, a large part of the first file is read into the dictionary. In yet another example, if the file size of every file in the sorted file list 204 is small and very few files share the same file group or file type, thereby indicating a low degree of between-file redundancy, then the dictionary size may be set at a relatively small size so that the speed of searching the dictionary improves.

In one embodiment, the dictionary size is initially set to a default dictionary size. The dictionary size is then increased if certain increment-triggering events are encountered during the examination of each file in the sorted file list 204. These events may include, for example, finding two files of the same file name and extension, finding two files of the same file extension, finding two files of the same file type, finding two files of the same file group, and so forth. Each time one of such increment-triggering events is encountered, the dictionary size is increased. Each of such increment-triggering events may be associated with the same or a different increment amount in dictionary size. Increment amount may be determined as percentages of the size of some or all of the files in the sorted file list 204, as percentages of the default dictionary size, as numbers of bytes, and so forth.

In another embodiment, the dictionary size is decreased if certain decrement-triggering events are encountered. In yet another embodiment, both increment-triggering events and decrement-triggering events may be used to adjust the dictionary size. The dictionary size may also be increased or decreased in proportion to the total size of files in the sorted file list 204 and the average file size.

The examination of the redundancy characteristics and the determination of the dictionary size may also be made at block 404, at the time the first-stage compression process examines the files of the original file list 202 and creates the sorted file list 204.

6. Intermediary Output File Is Formed

After the dictionary size is set and a dictionary method is applied to the unified input data file 302, at block 412 of FIGURE 4, the first-stage compression process forms the intermediary output file 312. One embodiment of the intermediary output file 312 includes two types of data: tokens of position/length pairs and uncompressed symbols. A token or an uncompressed symbol is accompanied by a one-bit 0/1 flag that indicates whether it is compressed or uncompressed. This embodiment of the intermediary output file 312 may be produced using LZSS or another dictionary method that produces uncompressed symbols and position/length pairs. After the intermediary output file 312 is produced, the first-stage compression process proceeds to an end state at block 414.

E. Second-Stage Compression

FIGURE 5 illustrates a flowchart of one embodiment of a second-stage compression process. The second-stage compression process begins with a start state at block 500 and proceeds to block 502. At block 502, the second-stage compression process compresses the file header portion 306 into the compressed file header portion 324. It should be understood that the first-stage compression process may also be used to compresses the file header portion 306. File header information may include, for example, text identifying the computer on which a file is located, the hard drive on which a file is located, the directory on which a file is located, and so forth. Because multiple files may be located on the same computer, the same hard drive or the same directory, the file header information may include identical sub-strings of substantial length. File header information may also include, for example, the date and time of file creation or modification,

and the file size. File header information for such data may also include redundant data. A compression method such as the Order-1 Dynamic Markov model may be suitable for compressing the file header portion 306. The Order-1 Dynamic Markov model uses one preceding symbol to predict (i.e., assign the probability of) the appearance of the next symbol, working dynamically by adjusting probabilities based on prior symbol pairing patterns in this section.

At block 504, the intermediary output file 312 is separated into multiple intermediary output sections. In one embodiment, the intermediary output file 312 is separated into four intermediary output sections: a first section for the 1-bit flags, a second section for uncompressed symbols, a third section for position codes, and a fourth section for length codes. As described in the paragraphs above, in this embodiment, the intermediary output file 312 includes uncompressed symbols and tokens of position/length pairs that are accompanied by a 1-bit flag indicating whether it is compressed. Other embodiments of multiple sections may also be used. For example, one embodiment separates the intermediary output file 312 into two sections, one section for uncompressed symbols and one section for the rest of the intermediary output data.

The second-stage compression process then applies one or more compression methods to the intermediary output sections to take advantage of the data characteristics in each section. The compression of the sections may be carried out in sequence or in parallel. The following paragraphs suggest some compression methods that may be suitable for each of the sections, although other compression methods may also be used. In addition, it may also become desirable to not compress some sections.

Still referring to FIGURE 5, at block 506, the second-stage compression process compresses the first section. The first section for the 1-bit flags is characterized by codes representing the binary 0/1 values. Because this section stores codes of binary values, a compression method such as the dynamic Huffman method or the RLE method may be suitable for compressing this section. The RLE (Run Length Encoding) method compresses data by replacing a string of identical symbols (called a run length) with one token. The token stores the symbol and a number representing the length of the run of identical symbols.

At block 508, the second-stage compression process compresses the second section. The second section for uncompressed symbols is characterized by codes representing symbols with no match in the dictionary and are thus uncompressed. Codes representing the "a-z" letters, the "A-Z" letters, and the 0-9 numbers may appear frequently in this section. Therefore, a compression method such as a Pseudo-Contextual method may be suitable for compressing this section. Using the Pseudo-Contextual method, each symbol is assigned to a group. Groups may include the "a-z" letter group, the "A-Z" letter group, and the 0-9 number group. In one embodiment, uncompressed symbols are assigned to up to sixteen groups. A symbol is represented by a group flag and a distance code. The group flag indicates the group to which the symbol is assigned. A distance code indicates the distance between the symbol and the first symbol of the group. For example, a symbol "A" assigned to the "A-Z" group has a distance code of 0, a symbol "C" assigned to the same group has a distance code of 2. Therefore, a symbol "A" in the "A-Z" group and a symbol "0" in the 0-9 group share a distance code

of 0. Symbols "A" and "0" are therefore redundant to some degree. The Pseudo-Contextual method then compresses the group flag-distance code pairs.

At block 510, the second-stage compression process compresses the third section. The third section for position codes is characterized by codes representing numbers. These numbers are typically small numbers and are typically less than an upper range such as one thousand or one million, depending on the dictionary size. Therefore, a compression method such as the dynamic Huffman method or the static Huffman method may be suitable for compressing this section.

In one embodiment, the second-stage compression process separates the position codes into multiple sub-sections and compresses the multiple sub-sections. For example, in one embodiment, the second-stage compression process stores the first eight bits of each position code in a first sub-section, and stores the rest of the bits of each position code in a second sub-section. Compared with the original position codes, the data in the first sub-section represent smaller numbers and are likely to have smaller variations. Therefore, compressing the position codes as multiple sub-sections may achieve better compression ratio than compressing the position codes as one section. It should be understood that the fourth section for length codes may also be compressed as multiple sub-sections.

At block 512, the second-stage compression process compresses the fourth section. The fourth section for length codes is characterized by codes representing numbers. These numbers are typically small numbers and are typically less than an upper range such as one hundred or one thousand depending on the dictionary size and the degree of redundancy. Therefore, a compression method such as the dynamic Huffman method or the static Huffman method may be suitable for compressing this section.

Still referring to FIGURE 5, at block 514 the second-stage compression process combines the compressed results from blocks 506, 508, 510 and 512 to form the final compressed data portion 326. The second-stage compression process combines the final compressed data portion 326, the archive header portion 304 and the compressed file header portion 324 to produce the final output file 322. In one embodiment, after the compressed file header portion 324 and the final compressed data portion 326 are produced, they are appended successively to the archive header portion 304.

In one embodiment, a recovery information portion 328 is appended to the final output file at block 516. The recovery information portion 328 includes recovery information such as the CRC about the final compressed data portion 326. In one embodiment the second stage compression process prompts a user to determine whether to protect the final output file 322 against damage by appending a recovery information portion 328. In one embodiment, the recovery information portion 328 is also compressed. One embodiment of the final output file 322 includes the file header portion 306 but not the compressed file header portion 324. In one embodiment the second-stage compression process updates the archive header portion 304. In one embodiment, the second-stage compression process combines the ZIP archive header 334, the ZIP central directory header 336 and the ZIP end of central directory header 338 with the final output file 322 to form the ZIP-recognized output file 332, which may be recognized by any utility program that supports the ZIP file format.

A utility program that recognizes the ZIP-recognized output file 332 may typically list information such as file names, sizes before compression, sizes after compression and compression ratios of the multiple files in the sorted file list 204, and so forth. The second-stage compression process then proceeds to an end state at block 518.

In another embodiment, the second-stage compression process is omitted; the archive header portion 304, the file header portion 306 (or the compressed file header portion 324 in another embodiment) the intermediary output file 312, and the recovery information portion 328 are combined to form the final output file 322.

Additional embodiments of a multiple-file compression system and processes are illustrated in Appendix A. Furthermore, Appendix B includes one embodiment of a set programming code that implements an embodiment of a multiple file compression system and processes. The disclosed program is written in the programming language C++, though it is recognized that in other embodiments, other languages may be used. It is recognized that the specification includes the description disclosed in Appendix A and Appendix B. Appendix A and Appendix B form a part of this specification.

III. CONCLUSION

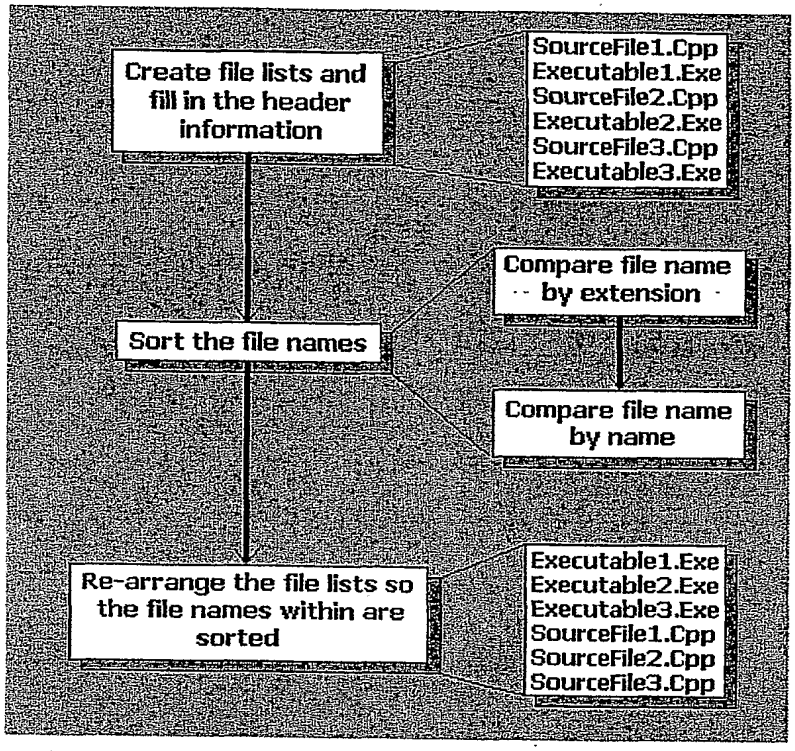
Although the foregoing has been a description and illustration of specific embodiments of the invention, various modifications and changes can be made thereto by persons skilled in the art, without departing from the scope and spirit of the invention as defined by the following claims. One embodiment of the claimed method is implemented in C++ computer programs, although other hardware, software, firmware and combinations may also be used for implementation. Accordingly, the breadth and the scope of the present invention should be defined only in accordance with the following claims and their equivalents.

APPENDIX A

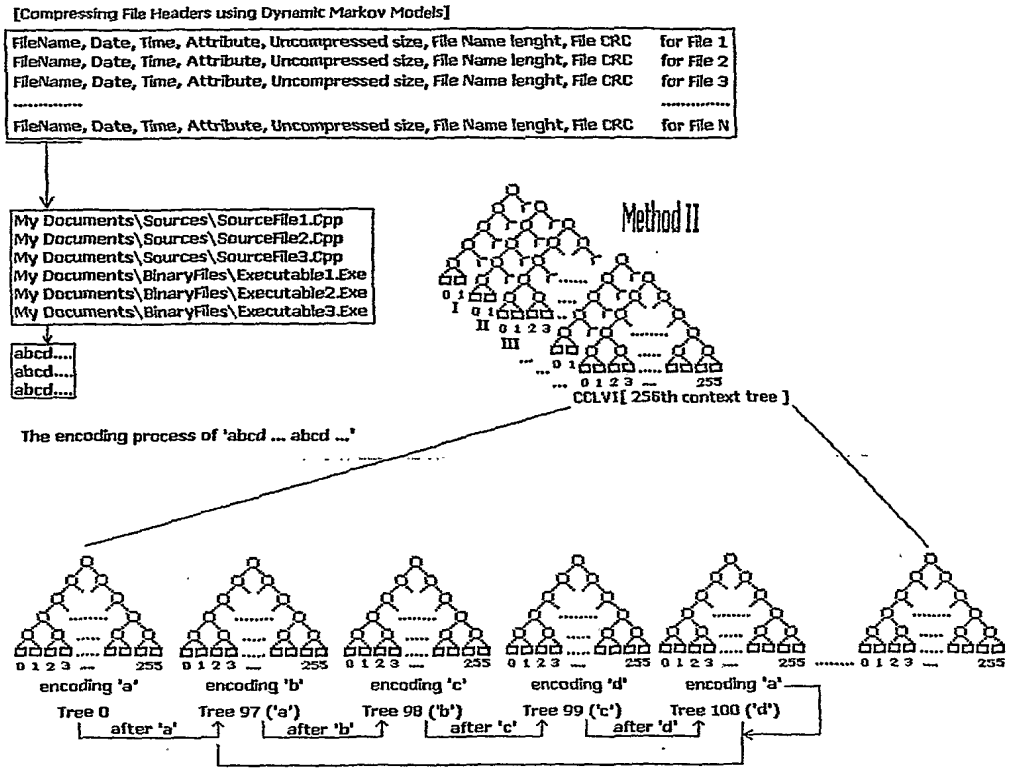
This appendix forms a part of the specification of the application entitled "SYSTEMSAND METHODS FOR MULTIPLE-FILE DATA COMPRESSION," internal reference number REALNET.136A

SYSTEM AND METHOD FOR DATA COMPRESSION

The present inventive was designed as a compression archiver, which gathers files to be compressed together and treats them as a contiguous stream of data. Files that are compressed are analyzed and sorted according to their extension and then the file name.



The sorting step is required and it is one of the most important aspects of the whole process, because the present invention takes advantage of between-file similarities and the compression ratio is improved whenever the redundancy is encountered in any form. Sorting files helps putting similar extension files together as they tend to have the same type of information within.



The output stream contains three sections. The first section is the archive header part. It contains the whole archive information, like creation time, whole archive CRC, flags for determining whether the archive is locked or damage protected or not. The second section is the header part, which contains all the headers and all the information required to uncompress and bring the files to their initial state, regardless the order they were compressed before. Please note that the order of decompressing files is different by the one of reading the files, because of the sorting process.

The third section is the compressed stream part. There are no flags to describe the beginning and the end of files, but only markers for blocks. The stream contains blocks and a block can be seen as a contiguous stream of bytes.

SECTION I
Archive Header

SECTION II
Header Blocks

SECTION III
Marker Block I -> Compressed Block I

Marker Block II -> Compressed Block II
Marker Block II -> Compressed Block II

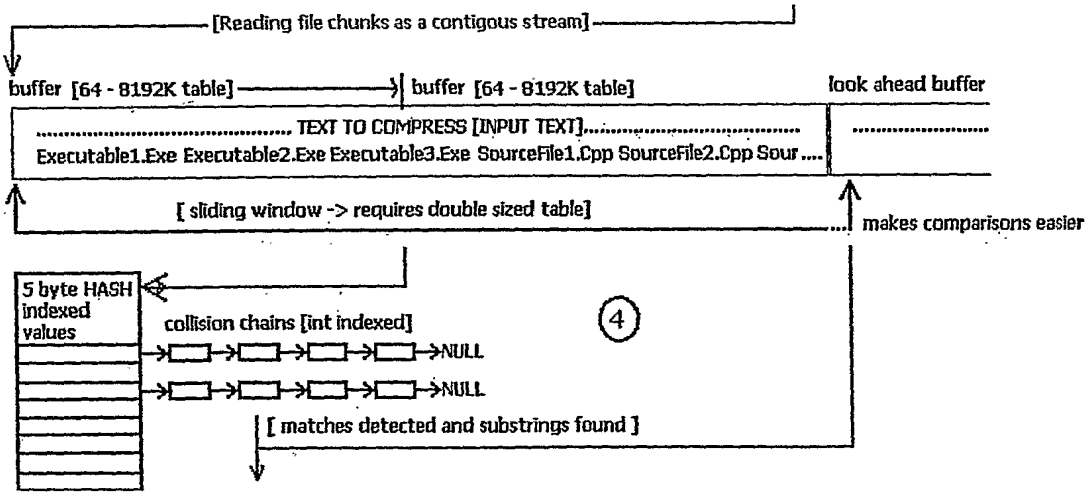
In some cases there is even section IV, which is related to the damage protection and contains blocks of information based upon CRCs (cyclic redundancy codes) performed on the compressed stream blocks.

SECTION IV
Marker Block I -> Information Recovery Block I
Marker Block II -> Information Recovery Block II
Marker Block II -> Information Recovery Block II

After the files are analyzed there are three major steps in the present inventive program.

The First step is file headers compression. There is only one block of information that contains the file headers and this block is compressed using Method II, described below. Outside the first section of XZIP archive, which is the actual header, there is nothing which remains uncompressed. The file headers and the compressed blocks contain compressed data or are compressed themselves.

The second step is the actual compression step, which implies a slightly modified LZ77 (called LZSS). The files previously sorted are read a dynamically allocated table (which can have different sizes ranging from 64K to 8192K) is filled with the bytes read from the files.

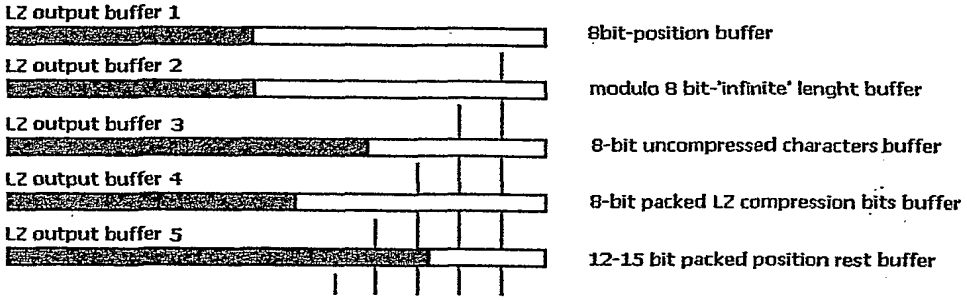


From the beginning of the table, which acts as a sliding window, meaning that always matches and sub-strings will be searched within the table size, even if the table pointer advances, because the table is doubled in size. Five bytes are hashed in two interlaced

four-byte keys, and then the result is compared to a 1024K table entry. If the table entry is not empty, then for every location that is chained in that entry sub-strings are searched and the biggest one is selected. However, a heuristic method is applied (Longest Frame First), and the sub-string is not selected if there is another one longer stating from the next character. When the sub-string with the longest length is selected, then the whole output will be made up of a pair of position and length.

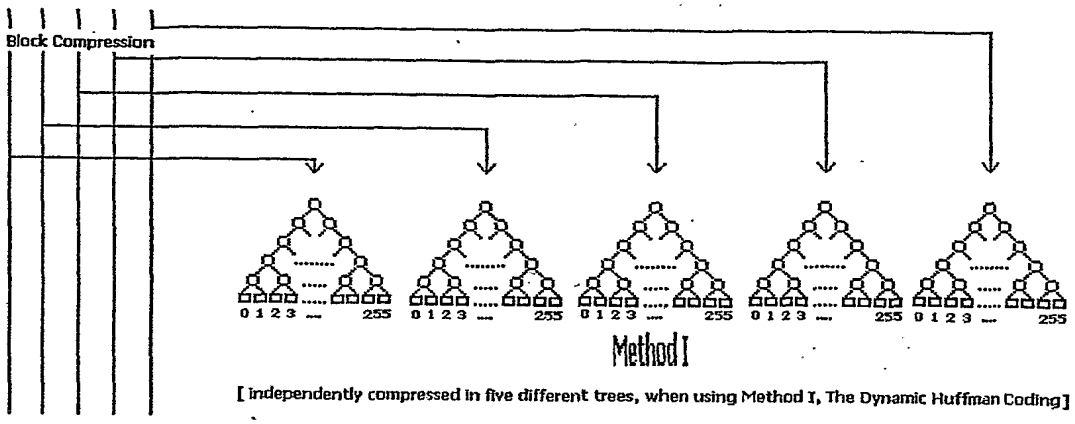
The whole output of this algorithm will contain basically uncompressed characters and position/length pairs, and 0/1 bits to determine which is which. The positions rely in the range of the table size from 64K to 8192K, which is from 16 bit to a 23 bit-number. This will be split in two different numbers, and they will fit in two different blocks. The former will contain the first 8 bits, and the latter will contain the rest, an 8 to 15 bit number.

The uncompressed characters will fit in a different block, and will not occupy the same block as the length bytes. This will made a total of 5 output blocks, if we number the bit block.



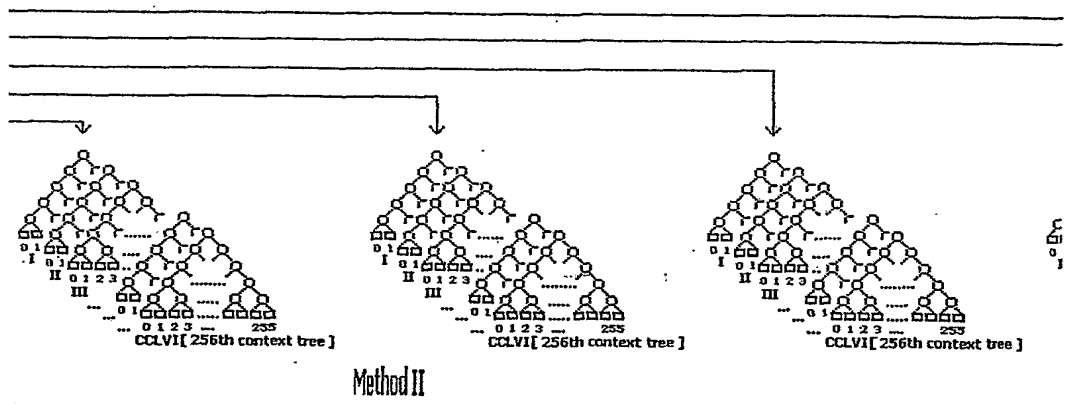
Block 1 – Bit block
Block 2 – Literal block (uncompressed bytes)
Block 3 – Length block (the sub-string length, can be unlimited and it's inserted modulo 255)
Block 4 – Distance block (the position of the sub-string which is kept as a distance from the reading pointer)
Block 5 – Rest block (the 8-15 bit number as a rest from the position number)

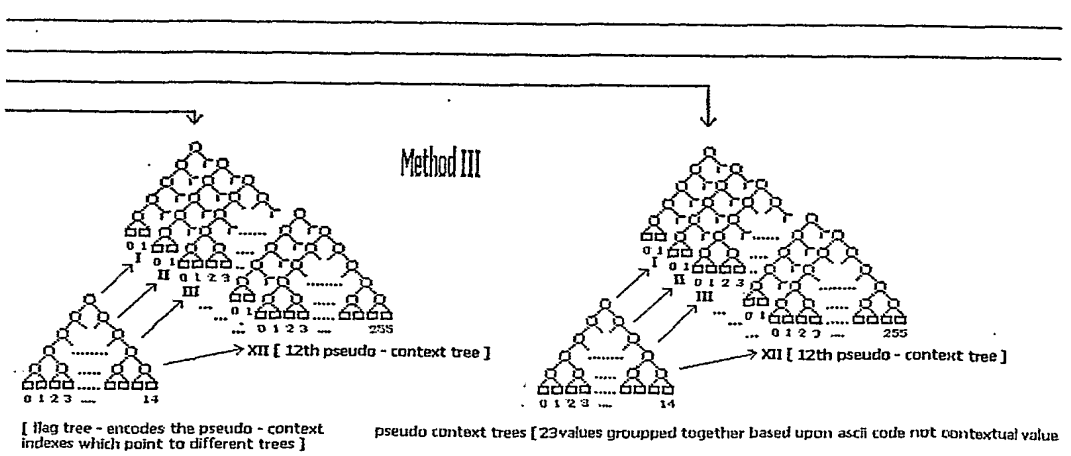
The third and the last step is the block compression which implies the Dynamic Huffman compression in three different ways. Two context techniques are applied, the Dynamic Markov Model (1-byte context)



and The Pseudo-Contextual Model (1 flag, 1byte context).

This with the simple Dynamic Huffman which can be called a 0-byte context make three different compression method that are applied on the LZ output blocks.





Please note that only one Method, Method I, II or III is applied on a buffer, but only Method II is applied on the file headers block, because of the 1-byte context technique.

The attached source illustrates one program in a series of separate files for implementing the present inventive system and method for data compression. The disclosed program is written in C++, although the present invention can be implemented in other languages as would be known to one of skill in the art.

APPENDIX B

This appendix forms a part of the specification of the application entitled "SYSTEMSAND METHODS FOR MULTIPLE-FILE DATA COMPRESSION," internal reference number REALNET.136A

```
/*************************************************************************
***********
 *
          *
 *       This is the user interface - console printing procedures
          *
 *
          *
 *          From Xtreme 106 (DLL) package by Sabin, Belu (c) 1999
          *
 *
          *
 *************************************************************************
***********/

#pragma pack (1)

#include <stdio.h>
#include <conio.h>
#include <windows.h>

#include "console.h"
#include "exports.h"

short  ratio;
extern XTREME106_CMDSTRUCTURE  MyXtrStructure;

void Printf(char *Text, char *delimiter, int atribute)
{
  if(MyXtrStructure.MinorStatusMessageDisplayFunction != NULL)
     MyXtrStructure.MinorStatusMessageDisplayFunction(Text, delimit
er, atribute);
}

void Print(char *Text1, char *delimiter1, int atribute1, char *Text
2, char *delimiter2, int atribute2,
           char *Text3, char *delimiter3, int atribute3, char *Text
4, char *delimiter4, int atribute4,
                  int   procent)
{

    if(MyXtrStructure.MajorStatusMessageDisplayFunction != NULL)
    {
       MyXtrStructure.MajorStatusMessageDisplayFunction(Text1, deli
miter1, atribute1,
```

Console

```
          Text2, delimiter2, atribute2, Text3, delimiter3, atribute3
, Text4, delimiter4, atribute4, procent);
    }
}

int ScanForAchar(char *mess, int but_no)  // a callback FileRename
function must be implemented !
{
   if(MyXtrStructure.Utility_Function  != NULL)
        return(MyXtrStructure.Utility_Function(mess, but_no));
   else
        return 'N'; // no as answer
}
```

console

```
int  ScanForAchar(char *mess, int but_no);
void Printf(char *Text, char *delimiter, int atribute);
void Print(char *Text1, char *delimiter1, int atribute1, char *Text
2, char *delimiter2, int atribute2,
           char *Text3, char *delimiter3, int atribute3, char *Text
4, char *delimiter4, int atribute4, int percent);
```

```
/**************************************************************************
***********
 *
          *
 *                          Contextual Encoding !
          *
 *              The best in literals encoding / decoding
          *
 *            From Xtreme 106 (DLL) package by Sabin, Belu
          *
 *
          *
 **************************************************************************
***********/
#pragma pack (1)

#include <string.h>
#include <stdlib.h>
#include <conio.h>
#include <assert.h>
#include <stdio.h>
#include <io.h>

#include "Util.h"
#include "Contextual.h"
#define MAX_READ    10240
#define MAX_MODELS  12

struct contextModel XXXContextualModel[MAX_MODELS];
unsigned int Interval_Low;
unsigned int Interval_Len;
unsigned int shifts, value;
unsigned short  buffer = 0, mask = 128;

//FILE  *infile;
//FILE  *outfile;

// extern unsigned.int  XXXInBufC;
//unsigned int  XXXOutBufC;
//unsigned char XXXInBuf [MAX_READ];
//unsigned char XXXOutBuf[MAX_READ * 2];

struct Fill_In_Data Fill[ MAX_MODELS + 2 ] =
{
        { 1,    0,      0,     16,   12}, { 1,    1,      0,    2,
  4},
```

Contextual

```
        { 0,    0,    2,    4,    4}, { 0,    0,    4,    8,
  4},
        { 0,    0,    8,   32,    4}, { 0,    0,   16,   32,
  3},
        { 0,    0,   48,   32,    3}, { 0,    0,   64,   32,
  3},
        { 0,    0,   96,   32,    3}, { 0,    0,  128,   32,
  2},
        { 0,    0,  160,   32,    2}, { 0,    0,  192,   32,
  2},
        { 0,    0,  224,    0,    0}, { 1,  255,    0,    0,
  0}
};

//unsigned int textsize, printcount, outcount, codesize;

#define MINIM(a,b) ((a<b) ? a : b)

inline int retrieveContextualEncodedCode(struct contextModel *XXX)
{
  register unsigned int TotalF;
  register unsigned int CummulatedLow, Low, CummulatedHigh, Range;
  register unsigned int ValuePerUnit,  END;
  register short symbol = 0;
  register unsigned short i;
  register unsigned short X = XXX->Step;

  CummulatedLow  = CummulatedHigh  = 0;
  TotalF         = XXX->TotalFrequency;
  ValuePerUnit   = Interval_Len / TotalF;
  END            = MINIM(TotalF - 1, value / ValuePerUnit);

  while(CummulatedHigh <= END)
   {
      symbol++;
      CummulatedHigh += XXX->Freq[symbol];
   }

  Range          = XXX->Freq[symbol];
  CummulatedLow = CummulatedHigh - Range;
  Low            = ValuePerUnit * CummulatedLow;
  value         -= Low;
  Interval_Len   = (CummulatedHigh < TotalF) ? ValuePerUnit * Range
  Interval_Len - Low;

  while(Interval_Len <= HALF)
```

Contextual

```
    {
      Interval_Len <<= 1;
      if ((mask >>= 1) == 0)
          {
        buffer = XXXInBuf[XXXInBufC++];
        mask = 128;
          }
       value = (value << 1) + ((buffer & mask) != 0);
    }

   XXX->Freq[symbol]       += X;
   XXX->TotalFrequency += X;

   if(XXX->TotalFrequency > XXX->MaximumFreq)
   {
     XXX->TotalFrequency = 0;
     for(i=1; i <= XXX->SymbolsNo; i++)
     {
       XXX->Freq[i]          = (XXX->Freq[i] + 1) >> 1;
       XXX->TotalFrequency += XXX->Freq[i];
     }
  }

   return (symbol);
}

void storeContextualEncodedCode(struct contextModel *contextModel,
 unsigned int symbol)
{
    register unsigned int i;
    register unsigned short X = contextModel->Step;
    register unsigned int TotalF;
    register unsigned int CummulatedLow, CummulatedHigh, Range, Low;
    register unsigned int ValuePerUnit;

    CummulatedLow = 0;
    TotalF         = contextModel->TotalFrequency;
    Range          = contextModel->Freq[symbol];
    ValuePerUnit   = Interval_Len / TotalF;
    for(i=1; i<symbol; i++)       CummulatedLow = CummulatedLow + con
textModel->Freq[i];
    CummulatedHigh = CummulatedLow + contextModel->Freq[symbol];
    Low = ValuePerUnit * CummulatedLow;
    Interval_Low +=  Low;
    Interval_Len   = (CummulatedHigh < TotalF) ? ValuePerUnit * Range
 : Interval_Len - Low;
```

Contextual

```
 // Renormalize -->
 //
 // 0                                HALF                                F
ULL
 // |--------------------------------|--------------------------------
-|
 //         Output bit 0                        Output bit 1
 //
 //             [.................]
 //             /\               /\
 //            /  \             /  \
 //  an Interval+CummulatedLow      /    \
 //        and an       Interval_Len
 //
 // Acest if imi lua aprope o secunda !!!!!
 // if(Interval_Len == 0)
 //    printf("\n\n Alarm, remormalization will fail ! \n");

 while(Interval_Len <= HALF) // everytime we are sending BITS - 2 b
its out
 {
   if(Interval_Low + Interval_Len <= FULL)
   {
        if ((mask >>= 1) == 0)
           {
         XXXOutBuf[XXXOutBufC++] = (unsigned char)buffer;
         buffer = 0;  mask = 128;
           }

          for ( ; shifts > 0; shifts--)
          {
         buffer |= mask;
         if ((mask >>= 1) == 0)
                {
           XXXOutBuf[XXXOutBufC++] = (unsigned char)buffer;
            buffer = 0;  mask = 128;
                }
          }
   }
   else
   if(FULL <= Interval_Low)
   {

          buffer |= mask;
```

Contextual

```
            if ((mask >>= 1) == 0)
            {
          XXXOutBuf[XXXOutBufC++] = (unsigned char)buffer;
          buffer = 0;  mask = 128;
            }

            for ( ; shifts > 0; shifts--)
            {
          if ((mask >>= 1) == 0)
                {
            XXXOutBuf[XXXOutBufC++] = (unsigned char)buffer;
            buffer = 0;  mask = 128;
                }
            }

            Interval_Low -= FULL; // this is equ with -= FULL (remov
ing the highest bit, 1)
    }
    else
    {
        shifts++;
        Interval_Low -= HALF; // this is equ with -= N2 (removing th
e highest bit, 1)
    }
    Interval_Low <<= 1;
    Interval_Len <<= 1;       // we are increasing the interval till
 it will go above HALF
  }

  contextModel->Freq[symbol]    += X;
  contextModel->TotalFrequency += X;
  if(contextModel->TotalFrequency > contextModel->MaximumFreq)
  {
    contextModel->TotalFrequency = 0;
    for(i=1; i <= contextModel->SymbolsNo; i++)
         {
        contextModel->Freq[i] = (contextModel->Freq[i] + 1) >> 1;
        contextModel->TotalFrequency += contextModel->Freq[i];
    }
  }


}

int getIndex(int n)
{
```

Contextual

```
    int val;

    if (n >= 255) val = 14;
    else
    if (n >= 224) val = 13;
    else
    if (n >= 192) val = 12;
    else
    if (n >= 160) val = 11;
    else
    if (n >= 128) val = 10;
    else
    if (n >=  96) val = 9;
    else
    if (n >=  64) val = 8;
    else
    if (n >=  48) val = 7;
    else
    if (n >=  16) val = 6;
    else
    if (n >=   8) val = 5;
    else
    if (n >=   4) val = 4;
    else
    if (n >=   2) val = 3;
    else
    if (n >=   1) val = 2;
    else          val = 1;

    return val;
}


void ContextualEncodeBuffer(unsigned int read, unsigned char *Inbuf
, unsigned char *Outbuf, unsigned int *out)
{
  unsigned int i;
  unsigned int j;

  for(i=0; i < MAX_MODELS; i++)
  {
    for(j=1; j <= Fill[i].symbolsNumber; j++) XXXContextualModel[ i
 ].Freq[j] = Fill[i].step;

    XXXContextualModel[ i ].TotalFrequency = Fill[i].step * Fill[i]
.symbolsNumber;
```

Contextual

```
    XXXContextualModel[ i ].Freq[0]          = XXXContextualModel[ i
].Freq[Fill[i].symbolsNumber + 1] = 0;
    XXXContextualModel[ i ].SymbolsNo        = Fill[i].symbolsNumber;
    XXXContextualModel[ i ].Step             = Fill[i].step;
    XXXContextualModel[ i ].MaximumFreq      = 1000;
  }

  mask = 128;
  value =  0;
  buffer = 0;
  Interval_Low = 0;
  Interval_Len = DOI_LA(BITS-1);

  XXXOutBufC = 0;
  for(i=0; i< read; i++)
  {
    int flag = getIndex( Inbuf[i] ) - 1;
    storeContextualEncodedCode(&XXXContextualModel[0], flag + 1);
    if(Fill [ flag  ].flag == 0)
                storeContextualEncodedCode(&XXXContextualModel[ fla
g - 1 ], Inbuf[i] - Fill [ flag ].thresholdValue + 1);

  }

  for (i=1; i<= BITS; i++)
  {
    register short b;
        register short bit = ((Interval_Low >> (BITS-i)) & 0x1);

    if(bit) buffer |= mask;
    if ((mask >>= 1) == 0)
        {
      XXXOutBuf[XXXOutBufC++] = (unsigned char)buffer;
      buffer = 0;  mask = 128;
        }

        for ( ; shifts > 0; shifts--)
        {
        b = !bit;
        if(b) buffer |= mask;
        if ((mask >>= 1) == 0)
                {
         XXXOutBuf[XXXOutBufC++] = (unsigned char)buffer;
         buffer = 0;  mask = 128;
                }
        }
```

Contextual

```
  }

  if(mask != 128) XXXOutBuf[XXXOutBufC++] = (unsigned char)buffer;

  memcpy(Outbuf, XXXOutBuf, XXXOutBufC);
  *out = XXXOutBufC;

}

void ContextualDecodeBuffer(unsigned int   read,   unsigned char *I
nbuf,
                        unsigned char *Outbuf, unsigned int  *OutNo)
{
  unsigned int x = read;
  unsigned int i;
  unsigned int j;

  for(i=0; i < MAX_MODELS; i++)
  {
    for(j=1; j <= Fill[i].symbolsNumber; j++)   XXXContextualModel[
 i ].Freq[j] = Fill[i].step;

    XXXContextualModel[ i ].TotalFrequency = Fill[i].step * Fill[i]
.symbolsNumber;
    XXXContextualModel[ i ].Freq[0]        = XXXContextualModel[ i
].Freq[Fill[i].symbolsNumber + 1] = 0;
    XXXContextualModel[ i ].SymbolsNo      = Fill[i].symbolsNumber;
    XXXContextualModel[ i ].Step           = Fill[i].step;
    XXXContextualModel[ i ].MaximumFreq    = 1000;
  }

  Interval_Low  = 0;
  Interval_Len = DOI_LA(BITS-1);

  XXXInBufC = XXXOutBufC = 0;

  memcpy(XXXInBuf, Inbuf, *OutNo);

  mask = 128;
  value =  0;
  buffer = XXXInBuf[XXXInBufC++];

  for(i=1; i < BITS; i++)
  {
    if ((mask >>= 1) == 0)
    {
```

Contextual

```
        buffer =  XXXInBuf[XXXInBufC++];
        mask = 128;
      }
      value = (value << 1) + ((buffer & mask) != 0);
    }
    while(x--)
    {
       int flag = (retrieveContextualEncodedCode(&XXXContextualModel[
0]) - 1) ;
       XXXOutBuf[XXXOutBufC++] = Fill [ flag  ].flag ? Fill [ flag ].
retValue :
                                 retrieveContextualEncodedCode(&XXXCo
ntextualModel[ flag - 1 ]) + Fill [ flag ].thresholdValue - 1;
    }

  memcpy(Outbuf, XXXOutBuf, XXXOutBufC);
  *OutNo = XXXOutBufC;
}
```

```
/**********************************************************************
***********
 *
          *
 *              This is the crc - utility draw used
          *
 *
          *
 *          From Xtreme 106 (DLL) package by Sabin, Belu (c) 1999
          *
 *
          *
 **********************************************************************
***********/

#pragma pack (1)

#include <stdlib.h>
#include "crc.h"

void updcrc(unsigned char *s, unsigned int n, unsigned int *whatcrc
) //pointer to bytes to pump through, number of bytes in s[] */
 {
    register unsigned int c;          /* temporary variable */

    c = *whatcrc;
    while(n--)
     {
        c = crc_32_tab[((int)c ^ (*s++)) & 0xff] ^ (c >> 8);
    };
    *whatcrc = (unsigned int)(c ^ INIT_CRC_VALUE);
 }

void local_short_crc(unsigned char *s, unsigned int n, unsigned sho
rt *whatcrc) //pointer to bytes to pump through, number of bytes in
 s[] */
{
   unsigned int O;
   O = INIT_CRC_VALUE;
   updcrc(s, n, &O); //pointer to bytes to pump through, number of
bytes in s[] */
   *whatcrc = (unsigned short)O;
}

void local_uint_crc(unsigned char *s, unsigned int n, unsigned int
*whatcrc) //pointer to bytes to pump through, number of bytes in s[
```

Crc

```
] */
{
   unsigned int O;
   O = INIT_CRC_VALUE;
   updcrc(s, n, &O); //pointer to bytes to pump through, number of
bytes in s[] */
   *whatcrc = (unsigned int)O;
}


unsigned int crc_32_tab[] =
{
  0x00000000L, 0x77073096L, 0xee0e612cL, 0x990951baL, 0x076dc419L,
  0x706af48fL, 0xe963a535L, 0x9e6495a3L, 0x0edb8832L, 0x79dcb8a4L,
  0xe0d5e91eL, 0x97d2d988L, 0x09b64c2bL, 0x7eb17cbdL, 0xe7b82d07L,
  0x90bf1d91L, 0x1db71064L, 0x6ab020f2L, 0xf3b97148L, 0x84be41deL,
  0x1adad47dL, 0x6ddde4ebL, 0xf4d4b551L, 0x83d385c7L, 0x136c9856L,
  0x646ba8c0L, 0xfd62f97aL, 0x8a65c9ecL, 0x14015c4fL, 0x63066cd9L,
  0xfa0f3d63L, 0x8d080df5L, 0x3b6e20c8L, 0x4c69105eL, 0xd56041e4L,
  0xa2677172L, 0x3c03e4d1L, 0x4b04d447L, 0xd20d85fdL, 0xa50ab56bL,
  0x35b5a8faL, 0x42b2986cL, 0xdbbbc9d6L, 0xacbcf940L, 0x32d86ce3L,
  0x45df5c75L, 0xdcd60dcfL, 0xabd13d59L, 0x26d930acL, 0x51de003aL,
  0xc8d75180L, 0xbfd06116L, 0x21b4f4b5L, 0x56b3c423L, 0xcfba9599L,
  0xb8bda50fL, 0x2802b89eL, 0x5f058808L, 0xc60cd9b2L, 0xb10be924L,
  0x2f6f7c87L, 0x58684c11L, 0xc1611dabL, 0xb6662d3dL, 0x76dc4190L,
  0x01db7106L, 0x98d220bcL, 0xefd5102aL, 0x71b18589L, 0x06b6b51fL,
  0x9fbfe4a5L, 0xe8b8d433L, 0x7807c9a2L, 0x0f00f934L, 0x9609a88eL,
  0xe10e9818L, 0x7f6a0dbbL, 0x086d3d2dL, 0x91646c97L, 0xe6635c01L,
  0x6b6b51f4L, 0x1c6c6162L, 0x856530d8L, 0xf262004eL, 0x6c0695edL,
  0x1b01a57bL, 0x8208f4c1L, 0xf50fc457L, 0x65b0d9c6L, 0x12b7e950L,
  0x8bbeb8eaL, 0xfcb9887cL, 0x62dd1ddfL, 0x15da2d49L, 0x8cd37cf3L,
  0xfbd44c65L, 0x4db26158L, 0x3ab551ceL, 0xa3bc0074L, 0xd4bb30e2L,
  0x4adfa541L, 0x3dd895d7L, 0xa4d1c46dL, 0xd3d6f4fbL, 0x4369e96aL,
  0x346ed9fcL, 0xad678846L, 0xda60b8d0L, 0x44042d73L, 0x33031de5L,
  0xaa0a4c5fL, 0xdd0d7cc9L, 0x5005713cL, 0x270241aaL, 0xbe0b1010L,
  0xc90c2086L, 0x5768b525L, 0x206f85b3L, 0xb966d409L, 0xce61e49fL,
  0x5edef90eL, 0x29d9c998L, 0xb0d09822L, 0xc7d7a8b4L, 0x59b33d17L,
  0x2eb40d81L, 0xb7bd5c3bL, 0xc0ba6cadL, 0xedb88320L, 0x9abfb3b6L,
  0x03b6e20cL, 0x74b1d29aL, 0xead54739L, 0x9dd277afL, 0x04db2615L,
  0x73dc1683L, 0xe3630b12L, 0x94643b84L, 0x0d6d6a3eL, 0x7a6a5aa8L,
  0xe40ecf0bL, 0x9309ff9dL, 0x0a00ae27L, 0x7d079eb1L, 0xf00f9344L,
  0x8708a3d2L, 0x1e01f268L, 0x6906c2feL, 0xf762575dL, 0x806567cbL,
  0x196c3671L, 0x6e6b06e7L, 0xfed41b76L, 0x89d32be0L, 0x10da7a5aL,
  0x67dd4accL, 0xf9b9df6fL, 0x8ebeeff9L, 0x17b7be43L, 0x60b08ed5L,
  0xd6d6a3e8L, 0xa1d1937eL, 0x38d8c2c4L, 0x4fdff252L, 0xd1bb67f1L,
  0xa6bc5767L, 0x3fb506ddL, 0x48b2364bL, 0xd80d2bdaL, 0xaf0a1b4cL,
```

Crc

```
    0x36034af6L, 0x41047a60L, 0xdf60efc3L, 0xa867df55L, 0x316e8eefL,
    0x4669be79L, 0xcb61b38cL, 0xbc66831aL, 0x256fd2a0L, 0x5268e236L,
    0xcc0c7795L, 0xbb0b4703L, 0x220216b9L, 0x5505262fL, 0xc5ba3bbeL,
    0xb2bd0b28L, 0x2bb45a92L, 0x5cb36a04L, 0xc2d7ffa7L, 0xb5d0cf31L,
    0x2cd99e8bL, 0x5bdeae1dL, 0x9b64c2b0L, 0xec63f226L, 0x756aa39cL,
    0x026d930aL, 0x9c0906a9L, 0xeb0e363fL, 0x72076785L, 0x05005713L,
    0x95bf4a82L, 0xe2b87a14L, 0x7bb12baeL, 0x0cb61b38L, 0x92d28e9bL,
    0xe5d5be0dL, 0x7cdcefb7L, 0x0bdbdf21L, 0x86d3d2d4L, 0xf1d4e242L,
    0x68ddb3f8L, 0x1fda836eL, 0x81be16cdL, 0xf6b9265bL, 0x6fb077e1L,
    0x18b74777L, 0x88085ae6L, 0xff0f6a70L, 0x66063bcaL, 0x11010b5cL,
    0x8f659effL, 0xf862ae69L, 0x616bffd3L, 0x166ccf45L, 0xa00ae278L,
    0xd70dd2eeL, 0x4e048354L, 0x3903b3c2L, 0xa7672661L, 0xd06016f7L,
    0x4969474dL, 0x3e6e77dbL, 0xaed16a4aL, 0xd9d65adcL, 0x40df0b66L,
    0x37d83bf0L, 0xa9bcae53L, 0xdebb9ec5L, 0x47b2cf7fL, 0x30b5ffe9L,
    0xbdbdf21cL, 0xcabac28aL, 0x53b39330L, 0x24b4a3a6L, 0xbad03605L,
    0xcdd70693L, 0x54de5729L, 0x23d967bfL, 0xb3667a2eL, 0xc4614ab8L,
    0x5d681b02L, 0x2a6f2b94L, 0xb40bbe37L, 0xc30c8ea1L, 0x5a05df1bL,
    0x2d02ef8dL
};
```

Defla32

```
/*************************************************************************
***********
 *
          *
 *          LZ 32, the powerful Xtreme 106 engine's engine room!
          *
 *
          *
 *          From Xtreme 106 (DLL) package by Sabin, Belu (c) 1999
          *
 *
          *
 *************************************************************************
***********/

#pragma pack (1)

#include  <stdio.h>
#include  <stdio.h>
#include  <string.h>

#include  "crc.h"
#include  "util.h"
#include  "solid.h"
#include  "trees.h"
#include  "deflate.h"
#include  "console.h"

extern char  TableIndex, Solida;
extern unsigned int  SolidK, SolidN;
extern short ratio;

#define max_insert_length  max_lazy_match
#define P_S *++scanning_pointer == *++matching_pointer
#define UPDATE_HASH(h,c)    (h = (((h)<<H_SHIFT) ^ (c)) & HASH_MASK)
#define HASH_IT(x)          (((x ^ (x >> 11))) & HASH_MASK)
#define HASHS(u1, u2)       (((u1 ^ (u2 >> 11))) & HASH_MASK)


#define PrintProgress  {short ratio;\
                        if(count_start > afis_start){ \
                        afis_start += 0x16000;\
                        if(Solida){\
                          if((SolidK != SolidN) || (SolidN == 1)) {
```

```
                                ratio = IntoarceProcent(count_start, (S
olida ? TotalSolidSize : TotalRead));\
                                sprintf(tmp, "%-60s", short_str(InFile,
 60));\
                                sprintf(tmp2,"  %3d.%d%c", ratio / 10,
ratio % 10, '%');\
                                Print(Repack ? " Repacking " : (ZipCloa
king ? " Zipping " : " Freezing "), "", ZipCloaking ? 14 : 3, tmp,
"", 2, tmp2, "", 3, "\r", "", 0, (ratio / 10)); } }\
                            else {\
                                ratio = IntoarceProcent(count_start, (S
olida ? TotalSolidSize : TotalRead));\
                                sprintf(tmp, "%-60s", short_str(InFile,
 60));\
                                sprintf(tmp2,"  %3d.%d%c", ratio / 10,
ratio % 10, '%');\
                                Print(Repack ? " Repacking " : (ZipCloa
king ? " Zipping " : " Freezing "),  "", ZipCloaking ? 14 : 3, tmp,
 "", 2, tmp2, "", 3, "\r", "", 0, (ratio / 10));} } }


void window_fill_in()
{
   register unsigned int i, j;
   unsigned int  some_extra, val;
   some_extra = (unsigned int)(window_size - (unsigned int)in_advan
ce_reading - (unsigned int)string_starting);
   if(some_extra == (unsigned)EOF)  some_extra--;
   else
   if (string_starting >= WSIZE+MAX_DIST)
    {
        memcpy((char *)Fereastra, (char *)Fereastra+WSIZE, (unsigne
d int)WSIZE);
        Solid_CRC = INIT_CRC_VALUE;
        updcrc((unsigned char *)Fereastra, (unsigned int)WSIZE, &So
lid_CRC);
        match_start      -= WSIZE;
        string_starting  -= WSIZE;
        for (i=0;i<HASH_SIZE;i++)
        {
           j=head[i].Li; j|=head[i].Hc << 16;
           val = ((unsigned int )(j >= WSIZE ? j-WSIZE : 0));
           head[i].Li = (unsigned short)(val);
           head[i].Hc = (unsigned  char)(val>>16);
        }
        for (i=0;i<WSIZE;i++)
```

```
        {
           j=prev[i].Li; j|=prev[i].Hc << 16;
           val = ((unsigned int )(j >= WSIZE ? j-WSIZE : 0));
           prev[i].Li = (unsigned short)(val);
           prev[i].Hc = (unsigned  char)(val>>16);
        }
        some_extra += WSIZE;
     }
   if (!EndOfFile)
     {
        i = read_from_file((unsigned char*)Fereastra+string_starting
+in_advance_reading, some_extra);
        if (i == 0 || i == (unsigned int  )EOF)   EndOfFile = 1;
        in_advance_reading += i;
     }
 }


int deflate()
{
    unsigned int  hash_head;                          // head of hash
 chain
    unsigned int  U1, U2, val;
    unsigned int  previous_matching_pointer;
// previous matching_pointer
    int  match_available = 0;                         // set if previ
ous matching_pointer exists
    register unsigned match_length = MIN_MATCH-1;     // length of be
st matching_pointer
    unsigned long K = 0;

    ratio = 0;
    Tree_Start();
    Print_Start();

    while (in_advance_reading != 0)
     {
        PrintProgress

            U1 = *(unsigned int *)(Fereastra + string_starting);
        U2 = *(unsigned int *)(Fereastra + string_starting + 1);
        hash_header = HASHS(U1, U2);
        hash_head =head[hash_header].Li;
        hash_head|=head[hash_header].Hc << 16;
        val = ((string_starting) & WMASK);
        prev[val].Li = (unsigned short)(hash_head);
```

```
        prev[val].Hc = (unsigned char)(hash_head>>16);
        head[hash_header].Li = (unsigned short)(string_starting);
        head[hash_header].Hc = (unsigned char)(string_starting>>16)
;
        prev_length = match_length, previous_matching_pointer = mat
ch_start;
        match_length = MIN_MATCH-1;
        if (hash_head != 0 && prev_length < max_lazy_match && strin
g_starting - hash_head <= MAX_DIST)
          {
              //match_length = tell_the_longest_match(hash_head);
              register int                    len;

              register int               best_len = prev_length;

              unsigned int              cur_match = hash_head;
              unsigned short        chain_length = max_chain_length
;                    // max hash chain length
              register unsigned char *scanning_pointer = Fereastra
+ string_starting;   // current string
              register unsigned char *matching_pointer;
                    // matched string
              unsigned int                    limit = string_starting
> (unsigned int )MAX_DIST ? string_starting - (unsigned int )MAX_DI
ST : 0;  //  nil  = 0
              register unsigned char     *strend = Fereastra + stri
ng_starting + MAX_MATCH;
              register unsigned char    scan_end = scanning_pointer
[best_len];
              register unsigned char   scan_end1 = scanning_pointer
[best_len-1];
              unsigned int val;
              char cond1, cond2;

              if (prev_length >= good_match)  chain_length >>= 2;

              for(;;)
              {
                matching_pointer = Fereastra + cur_match;

                // This is a method of getting rid quickly of match
es that are alledgedly bigger
                // than the current one !
                // We can really get rid very quickly of a matching
_pointer by checking it's boundary, only !
```

```
                if(best_len >= 5)
                {
                  if(matching_pointer[best_len] != scan_end ||
                   matching_pointer[best_len-1] != scan_end1) goto
Check;
                }

                if(*(unsigned int *)(matching_pointer) != *(unsigne
d int *)(scanning_pointer))    goto Check;
                if(*(matching_pointer+4) != *(scanning_pointer+4))
                              goto Check;

                scanning_pointer+=4;
                matching_pointer+=4;

                do{}while (P_S && P_S && P_S && P_S && P_S && P_S &
& P_S && P_S && scanning_pointer < strend);
                len = MAX_MATCH - (int )(strend - scanning_pointer)
;
                scanning_pointer = strend - MAX_MATCH;

                if (len > best_len)
                {
                   match_start = cur_match;
                   best_len = len;
                   if (len >= nice_match) break;
                   scan_end1  = scanning_pointer[best_len-1];
                   scan_end   = scanning_pointer[best_len];
                }
                Check : val = cur_match & WMASK;
                cur_match  = prev[val].Li;
                cur_match |= prev[val].Hc << 16;
                cond1 = cur_match > limit;
                cond2 = (cond1 ? --chain_length != 0 : 0);
                if(cond1 == 0 || cond2 == 0) break;
              }
              //  return best_len;
              match_length = best_len;

              if (match_length > in_advance_reading) match_length =
 in_advance_reading;
              if (match_length == MIN_MATCH && string_starting-matc
h_start > TOO_DISTANT) match_length--;
        }
        if (prev_length >= MIN_MATCH && match_length <= prev_length
```

```
          {
            int distance = string_starting - previous_matching_poin
ter - (MIN_MATCH - (MIN_MATCH == 4 ? 2 : 1));
            int len = prev_length - MIN_MATCH;
            in_advance_reading -= prev_length-1;
            prev_length -=  2;
            do {
                 string_starting ++;
                 count_start++;
                 U1 = *(unsigned int *)(Fereastra + string_starting
);
                 U2 = *(unsigned int *)(Fereastra + string_starting
 + 1);
                 hash_header = HASHS(U1, U2);
                 hash_head=head[hash_header].Li;
                 hash_head|=head[hash_header].Hc << 16;
                 val = ((string_starting) & WMASK);
                 prev[val].Li = (unsigned short)(hash_head);
                 prev[val].Hc = (unsigned char)(hash_head>>16);
                 head[hash_header].Li = (unsigned short)(string_sta
rting);
                 head[hash_header].Hc = (unsigned char)(string_star
ting>>16);
            } while (--prev_length != 0);
            match_available = 0;
            match_length = MIN_MATCH-1;
            string_starting++;
            count_start++;
            Tree_Encode(distance + 1, len);
        }
        else
         if (match_available)
          {
            Tree_Encode(0, Fereastra[string_starting-1]);
            string_starting++;
            count_start++;
            in_advance_reading--;
          }
         else
          {
            match_available = 1;
            string_starting++;
            in_advance_reading--;
            count_start++;
          }
    while(EndOfFile == 0 && in_advance_reading < MIN_LOOKAHEAD) win
```

Defla32

```
dow_fill_in();
  }
  if (match_available)
  {
      Tree_Encode(0, Fereastra[string_starting-1]);
  }

  Tree_Flush();
  PrintProgress

  return 1;
}

int inflate()
{
   int Dis=0, Pos=0, Len=0, Len2=0, stop = 0;
   unsigned int  CurPos = 0, i=0, j=0, k=0;
   unsigned int  flags = 0;

   for(;Read_Block() == 0;)
    {
      if(restbits > 8)
      {
        restbits-=9;
        stop = 1;
      }
      else stop = 0;
      for (i=0; i < BitBufC; i++)
      {
        flags = BitBuf[i];
        k = (i == BitBufC - 1? (restbits == 0 ? 8 : restbits)   : 8)
;
        for(j=0; j < k; j++)
         {
           if (flags & 1)
           {
            Fereastra[CurPos++] = LitBuf[LitBufC++];
            if (CurPos == WSIZE)
             {
               flush_window(CurPos);
               CurPos = 0;
             }
           }
          else
           {
              Dis    =  DistBuf[DistBufC++];
```

Defla32

```
            Dis  <<= D_REST_BITS;
            Dis  |=  (RestBuf2[RestBufC2++] << (D_REST_BITS - 8));
            Dis  |=  (RestBuf1[RestBufC1++] & 0xFF);
            Pos  = (CurPos - Dis - (MIN_MATCH - (MIN_MATCH == 4 ?
3 : 2))) & (WSIZE-1);
            Len  = LenBuf[LenBufC++];
            if (Len == 255)
            {
              Len  = LenBuf[LenBufC++];
              Len2 = LenBuf[LenBufC++];

              Len |= ((Len2 & 0x00ff) << 8);
            }
            Len += MIN_MATCH;
        E5:
            Fereastra[CurPos++] = Fereastra[Pos];
            Pos = (Pos + 1) & (WSIZE-1);
            if (CurPos == WSIZE)
             {
              flush_window(CurPos);
              CurPos = 0;
            }
           if(--Len != 0)  goto E5;
          }
          flags >>= 1;
      }
    }
      if(stop) break;
  }
 flush_window(CurPos);

 sprintf(tmp, " %-63s ", InFile);
 sprintf(tmp2, "%s", (TotalSum ? "not OK" : "Ok"));
 if(Solida == 0 )
 {
    Print(tmp," ", 2, "melted", " ", 3, Solida == 0 ? tmp2 : "
", "\n", 4, "", "", 0, -1);
 }
 return 0;
}
```

huffman

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "huffman.h"

#define  STATES_NO 1

FILE  *infile, *outfile;

short x;
unsigned int  outcount=0;
unsigned char  inbitstringlen = 0;
unsigned int   inbitstringbuf = 0;
unsigned int   outbitstringbuf = 0;
unsigned char  outbitstringlen = 0;

struct HuffmanTree GTree;// GallagerTree[STATES_NO];


void EncodeBuffer(unsigned int Size, unsigned char *Buffer, unsigne
d char *OutBuff, unsigned int *outcnt)
{
   unsigned int code;
   unsigned int code_buff;
   unsigned int code_length, k, k2;
   int temp_freq;
   int son_index, sibling_index, dad_code, sibling_code;
   int dad1_index, dad2_index, dad_index;


   (*outcnt)=0;
   outbitstringlen = 0;
   outbitstringbuf = 0;

   for (code_buff = 0; code_buff < MAX_CHARACTERS; code_buff++)
     {
       GTree.Spl_Freq[code_buff] = 1;
       GTree.Spl_Son[code_buff] = code_buff + ROOTPLUSONE;
       GTree.Spl_Dad[code_buff + ROOTPLUSONE] = code_buff;
     }

   for(code_buff = 0, code_length = MAX_CHARACTERS; code_length <=
```

huffman

```
ROOT; code_buff += 2, code_length++)
   {
       GTree.Spl_Freq[code_length] = GTree.Spl_Freq [code_buff] + G
Tree.Spl_Freq[code_buff + 1];
       GTree.Spl_Son [code_length] = code_buff;
       GTree.Spl_Dad [code_buff + 1] = code_length;
       GTree.Spl_Dad [code_buff] = GTree.Spl_Dad [code_buff + 1];
   }

   GTree.Spl_Freq[ROOTPLUSONE] = 65535;
   GTree.Spl_Dad[ROOT] = 0;

   for(k2 = 0; k2 < Size; k2++)
    {
      code = Buffer[k2];
      code_buff = 0;
          code_length = 0;
      k = GTree.Spl_Dad[code + ROOTPLUSONE];

      for(;;)
          {
        code_buff >>= 1;
        code_buff += (k & 1) ? MAX_FREQ : 0;
        code_length++;
            k = GTree.Spl_Dad[k];
                if(k == ROOT) break;
      }


      outbitstringbuf |= code_buff >> outbitstringlen;
      if ((outbitstringlen += code_length) >= 8)
      {
         OutBuff[(*outcnt)++] = (unsigned char)(outbitstringbuf >>
8);
                outbitstringlen -= 8;

         if (outbitstringlen >= 8)

            OutBuff[(*outcnt)++] = (unsigned char)(outbitstringbuf)
;
            outbitstringlen -= 8;
            outbitstringbuf = code_buff << (code_length - outbitstr
inglen);
         }
         else
          {
```

huffman

```
            outbitstringbuf <<= 8;
                }
        }

      if (GTree.Spl_Freq[ROOT] == MAX_FREQ)
          {
        dad2_index = 0;
        for (dad1_index = 0; dad1_index < ROOTPLUSONE; dad1_index++
)
                {
          if (GTree.Spl_Son[dad1_index] >= ROOTPLUSONE)
                {
            GTree.Spl_Freq[dad2_index] = (GTree.Spl_Freq[dad1_index
] + 1) >> 1;
            GTree.Spl_Son [dad2_index++] = GTree.Spl_Son[dad1_index
];
                }
              }

        for (dad1_index = 0, dad2_index = MAX_CHARACTERS; dad2_inde
x < ROOTPLUSONE; dad1_index += 2, dad2_index++)
              {
          dad_index = dad1_index + 1;
          GTree.Spl_Freq[dad2_index] = GTree.Spl_Freq[dad1_index] +
 GTree.Spl_Freq[dad_index];
            temp_freq = GTree.Spl_Freq[dad2_index];
          dad_index = dad2_index - 1;
            for (dad_index; GTree.Spl_Freq[dad_index] > temp_freq
; dad_index--); dad_index++;
            for(int u = dad2_index - dad_index - 1; u >= 0 ;  u--
) GTree.Spl_Freq[dad_index + u + 1] = GTree.Spl_Freq[dad_index + u]
;    GTree.Spl_Freq[dad_index] = temp_freq;
            for(int uu = dad2_index - dad_index - 1; uu >= 0 ;  u
u--) GTree.Spl_Son[dad_index + uu + 1] = GTree.Spl_Son[dad_index +
uu]; GTree.Spl_Son[dad_index] = dad1_index;
              }

        for (dad1_index = 0; dad1_index < ROOTPLUSONE; dad1_index++
)
              {
            dad_index = GTree.Spl_Son[dad1_index];
          if (dad_index >= ROOTPLUSONE) GTree.Spl_Dad[dad_index] =
dad1_index;
          else
                {
            GTree.Spl_Dad[dad_index + 1] = dad1_index;
```

```
                GTree.Spl_Dad[dad_index] = GTree.Spl_Dad[dad_index
+ 1];
                  }
                }
          }

      code = GTree.Spl_Dad[code + ROOTPLUSONE];

      for(;;)
          {
        GTree.Spl_Freq[code]++;
        dad_code = GTree.Spl_Freq[code];
        sibling_code = code + 1;

        if(GTree.Spl_Freq[sibling_code] < dad_code)
                {
              for(;;)
                  {
            sibling_code ++;
            if(dad_code <= GTree.Spl_Freq[sibling_code])
                        {
                       sibling_code--;
             break;
                        }
                  }

              GTree.Spl_Freq[code] = GTree.Spl_Freq[sibling_code];
          GTree.Spl_Freq[sibling_code] = dad_code;
          son_index = GTree.Spl_Son[code];
          GTree.Spl_Dad[son_index] = sibling_code;
          if (son_index < ROOTPLUSONE) GTree.Spl_Dad[son_index + 1]
 = sibling_code;
          sibling_index = GTree.Spl_Son[sibling_code];
          GTree.Spl_Son[sibling_code] = son_index;
          GTree.Spl_Dad[sibling_index] = code;
          if (sibling_index < ROOTPLUSONE) GTree.Spl_Dad[sibling_in
dex + 1] = code;
          GTree.Spl_Son[code] = sibling_index;
          code = sibling_code;
                }
           code = GTree.Spl_Dad[code];
           if(code == 0) break;
          }
   }

   if (outbitstringlen)
```

huffman

```
    {
      OutBuff[(*outcnt)++] = (unsigned char)(outbitstringbuf >> 8);
    }


}

void DecodeBuffer(unsigned int Size, unsigned char *Buffer, unsigne
d char *OutBuff, unsigned int *outcnt)
{
  int tmp1, tmp2, code;
    unsigned int incontor = 0;
   int temp_freq;
   int son_index, sibling_index, dad_code, sibling_code;
   int dad1_index, dad2_index, dad_index;


    (*outcnt) = 0;
    inbitstringlen = 0;
    inbitstringbuf = 0;

    for (tmp1 = 0; tmp1 < MAX_CHARACTERS; tmp1++)
    {
       GTree.Spl_Freq[tmp1] = 1;
       GTree.Spl_Son[tmp1] = tmp1 + ROOTPLUSONE;
       GTree.Spl_Dad[tmp1 + ROOTPLUSONE] = tmp1;
    }

   for(tmp1 = 0, tmp2 = MAX_CHARACTERS; tmp2 <= ROOT; tmp1 += 2, tm
p2++)
   {
       GTree.Spl_Freq[tmp2] = GTree.Spl_Freq [tmp1] + GTree.Spl_Fre
q[tmp1 + 1];
       GTree.Spl_Son [tmp2] = tmp1;
       GTree.Spl_Dad [tmp1 + 1] = tmp2;
       GTree.Spl_Dad [tmp1] = GTree.Spl_Dad [tmp1 + 1];
   }

   GTree.Spl_Freq[ROOTPLUSONE] = 65535;
   GTree.Spl_Dad[ROOT] = 0;

   for (unsigned int k = 0; k < Size; k++ )
   {
     code = GTree.Spl_Son[ROOT];

     for(;;)
```

huffman

```
    {
                if(code >= ROOTPLUSONE) break;
                for(;;)
                {
            if(inbitstringlen > 8) break;
            x = Buffer[incontor++];
            inbitstringbuf |= x << (8 - inbitstringlen);
            inbitstringlen += 8;
                }
          x = inbitstringbuf;
          inbitstringbuf <<= 1;
          inbitstringlen--;

          code += (x < 0);
          code = GTree.Spl_Son[code];
    }

    code -= ROOTPLUSONE;

          int xcode = code;

   if (GTree.Spl_Freq[ROOT] == MAX_FREQ)
        {
     dad2_index = 0;
     for (dad1_index = 0; dad1_index < ROOTPLUSONE; dad1_index++)

          {
       if (GTree.Spl_Son[dad1_index] >= ROOTPLUSONE)
              {
           GTree.Spl_Freq[dad2_index] = (GTree.Spl_Freq[dad1_index
] + 1) >> 1;
           GTree.Spl_Son [dad2_index++] = GTree.Spl_Son[dad1_index
];
              }
          }

     for (dad1_index = 0, dad2_index = MAX_CHARACTERS; dad2_index
 < ROOTPLUSONE; dad1_index += 2, dad2_index++)
          {
       dad_index = dad1_index + 1;
       GTree.Spl_Freq[dad2_index] = GTree.Spl_Freq[dad1_index] +
GTree.Spl_Freq[dad_index];
           temp_freq = GTree.Spl_Freq[dad2_index];
       dad_index = dad2_index - 1;
           for (dad_index; GTree.Spl_Freq[dad_index] > temp_freq;
 dad_index--); dad_index++;
```

huffman

```
            for(int u = dad2_index - dad_index - 1; u >= 0 ;  u--)
 GTree.Spl_Freq[dad_index + u + 1] = GTree.Spl_Freq[dad_index + u];
   GTree.Spl_Freq[dad_index] = temp_freq;
            for(int uu = dad2_index - dad_index - 1; uu >= 0 ;  uu
--) GTree.Spl_Son[dad_index + uu + 1] = GTree.Spl_Son[dad_index + u
u]; GTree.Spl_Son[dad_index] = dad1_index;
          }

      for (dad1_index = 0; dad1_index < ROOTPLUSONE; dad1_index++)

          {
            dad_index = GTree.Spl_Son[dad1_index];
        if (dad_index >= ROOTPLUSONE) GTree.Spl_Dad[dad_index] = d
ad1_index;
        else
                {
          GTree.Spl_Dad[dad_index + 1] = dad1_index;
              GTree.Spl_Dad[dad_index] = GTree.Spl_Dad[dad_index +
1];
                }
          }
        }

    code = GTree.Spl_Dad[code + ROOTPLUSONE];

    for(;;)
        {
      GTree.Spl_Freq[code]++;
      dad_code = GTree.Spl_Freq[code];
      sibling_code = code + 1;

      if(GTree.Spl_Freq[sibling_code] < dad_code)
          {
            for(;;)
                {
           sibling_code ++;
           if(dad_code <= GTree.Spl_Freq[sibling_code])
                        {
                          sibling_code--;
            break;
                        }
                }

           GTree.Spl_Freq[code] = GTree.Spl_Freq[sibling_code];
       GTree.Spl_Freq[sibling_code] = dad_code;
```

huffman

```
            son_index = GTree.Spl_Son[code];
            GTree.Spl_Dad[son_index] = sibling_code;
            if (son_index < ROOTPLUSONE) GTree.Spl_Dad[son_index + 1]
= sibling_code;

            sibling_index = GTree.Spl_Son[sibling_code];
            GTree.Spl_Son[sibling_code] = son_index;

            GTree.Spl_Dad[sibling_index] = code;
            if (sibling_index < ROOTPLUSONE) GTree.Spl_Dad[sibling_ind
ex + 1] = code;
            GTree.Spl_Son[code] = sibling_index;

            code = sibling_code;
                }

             code = GTree.Spl_Dad[code];
             if(code == 0) break;
            }

       OutBuff[(*outcnt)++] = (unsigned char)(xcode);
    }


}
unsigned char InBuf[MAX_READ];
unsigned char OutBuf[MAX_READ*2];
```

```
/*****************************************************************
***********
 *
          *
 *                Obsolete ZIP header comversion procedures
          *
 *
          *
 *          From Xtreme 106 (DLL) package by Sabin, Belu (c) 1999
          *
 *
          *
 *****************************************************************
***********/

#pragma pack (1)

#include "util.h"
#include "solid.h"
#include "string.h"
#include "zipheads.h"
#include "console.h"

ZIP_Header                                ZIP_H, ZIP_1;
ZIP_Central_Directory_Header            ZIP_CDH;
ZIP_End_Of_Central_Directory_Header  ZIP_CDHE;

int WriteFirstZipHeader(void)
{
    // create a Zip Header. The zip archives do not have a special
archive header, so
    // we will simulate

    char tmp___[_MAX_PATH];

    if(SolidArray[0].RealName[1] == ':' && SolidArray[0].RealName[2
] == '\\') // the disk is in front
    {
      memset(tmp___, 0, sizeof(tmp___));
      memcpy((unsigned char *)&tmp___[0], (unsigned char *)&SolidAr
ray[0].RealName[3], strlen(SolidArray[0].RealName) - 3);
    }
    else
      memcpy((unsigned char *)&tmp___[0], (unsigned char *)&SolidAr
ray[0].RealName[0], strlen(SolidArray[0].RealName));
```

```
    ZIP_H.signature = 0x04034b50;
    ZIP_H.version = 106;
    ZIP_H.general_bit = 2;
    ZIP_H.compression_method = 11;
    ZIP_H.time_ = 0;
    ZIP_H.date_ = 0;
    ZIP_H.crc32 = 0;
    ZIP_H.compressed   = 0;
    ZIP_H.uncompressed = SolidArray[0].size;
    ZIP_H.filenamel    = strlen(tmp___);
    ZIP_H.extra_len    = sizeof(struct ARH_Header);

    if(fwrite(&ZIP_H, 1, sizeof(ZIP_H), F2) != sizeof(ZIP_H))  Erro
r(7, "");
    if(fwrite(&tmp___, 1, ZIP_H.filenamel, F2) != ZIP_H.filenamel)
 Error(7, "");

    //if(fwrite(&ARH_H, 1, sizeof(struct ARH_Header), F2) != sizeof
(struct ARH_Header))  Error(7, "");
    if(FWrite_ARH_Header(&ARH_H, F2)) Error(7, ""); // rewritten, h
as a 4 byte CRC built in!


    memcpy((unsigned char *)&ZIP_1, (unsigned char *)&ZIP_H, sizeof
(ZIP_H));
    return 0;
}

int  WriteZipHeaders(long *AllWritten)
{
  long where_to;
  char tmp___[_MAX_PATH];
  unsigned int i, Extra = 0, Relative = 0;


  Print(" ", "", 14, " Updating Zip Header ...", " ", 14, " ", "\n"
, 14, "", "",  14, -1);

  ZIP_1.compressed = *AllWritten;
  where_to = ftell(F2);
  rewind(F2);
  if(fwrite(&ZIP_1, 1, sizeof(ZIP_1), F2) != sizeof(ZIP_1))  Error(
7, "");
  fseek(F2, where_to, SEEK_SET);
```

```
  for(i =0; i<SolidN; i++)
  {
    if(SolidArray[i].RealName[1] == ':' && SolidArray[i].RealName[2
] == '\\') // the disk is in front
    {
      memset(tmp___, 0, sizeof(tmp___));
      memcpy((unsigned char *)&tmp___[0], (unsigned char *)&SolidAr
ray[i].RealName[3], strlen(SolidArray[i].RealName) - 3);
      memset(SolidArray[i].RealName, 0, sizeof(SolidArray[i].RealNa
me));
      memcpy((unsigned char *)&SolidArray[i].RealName[0], (unsigned
 char *)&tmp___[0],  strlen(tmp___));
    }
  }

  for(i = 1; i<SolidN; i++) // first the ZIP_H headers
  {
    ZIP_H.signature    = 0x04034b50;
    ZIP_H.version      = 106;
    ZIP_H.general_bit = 2;
    ZIP_H.compression_method = 4;//(SolidArray[i].size < 40 ? 0 : (
(SolidArray[i].size % 10) + 1));
    ZIP_H.time_ = (unsigned short)(SolidArray[i].time_date);

    ZIP_H.date_ = (unsigned short)(SolidArray[i].time_date >> 16);

    ZIP_H.crc32         = 0xEA8CDEF6 ^ ((unsigned long)(SolidArray[i
].size * 991111));
    ZIP_H.compressed    = 0;
    ZIP_H.uncompressed = SolidArray[i].size;
    ZIP_H.filenamel     = strlen(SolidArray[i].RealName);
    ZIP_H.extra_len     = 0;

    if(fwrite(&ZIP_H, 1, sizeof(ZIP_H), F2) != sizeof(ZIP_H))  Erro
r(7, "");
    if(fwrite(&SolidArray[i].RealName, 1, ZIP_H.filenamel, F2) != Z
IP_H.filenamel)  Error(7, "");
  }

  where_to = ftell(F2);

  for(i =0; i<SolidN; i++)   // then the other ones !
  {
     ZIP_CDH.signature      = 0x02014b50;
     ZIP_CDH.ver            = 2580;
     ZIP_CDH.ver_extr       = 106;
```

Zipheads

```
      ZIP_CDH.general_bit    = 2;
      ZIP_CDH.compress_m     = 4;//(SolidArray[i].size < 40 ? 0 : ((S
olidArray[i].size % 10) + 1)); // de schepsis
      ZIP_CDH.time_date      = SolidArray[i].time_date;
      ZIP_CDH.crc32          = (i == 0 ? 0 : 0xEA8CDEF6 ^ ((unsigned
long)(SolidArray[i].size * 991111)));
      ZIP_CDH.compressed     = (i == 0 ? ZIP_1.compressed : 0);
      ZIP_CDH.uncompressed   = SolidArray[i].size;
      ZIP_CDH.file_n         = strlen(SolidArray[i].RealName);

      ZIP_CDH.extra_field    = (i == 0 ? ZIP_1.extra_len : 0);
      ZIP_CDH.external_attr  = 32;
      ZIP_CDH.file_comment   = 0;
      ZIP_CDH.relat_offset   = Relative;

      Relative += ZIP_CDH.compressed + sizeof(ZIP_H) + ZIP_CDH.file_
n +
                  ZIP_CDH.extra_field;

      if(fwrite(&ZIP_CDH, 1, sizeof(ZIP_CDH), F2) != sizeof(ZIP_CDH)
)  Error(7, "");
      if(fwrite(&SolidArray[i].RealName, 1, ZIP_CDH.file_n , F2) !=
ZIP_CDH.file_n)  Error(7, "");

      //if(i == 0) fwrite(&ARH_H, 1, sizeof(struct ARH_Header), F2);
            if(i == 0) FWrite_ARH_Header(&ARH_H, F2); // rewritten,
has a 4 byte CRC built in!

      Extra +=  sizeof(ZIP_CDH) + ZIP_CDH.extra_field + ZIP_CDH.file
_n;
    }

   ZIP_CDHE.signat_end  = 0x06054b50;
   ZIP_CDHE.total_entries_here = SolidN;
   ZIP_CDHE.total_entries      = SolidN;
   ZIP_CDHE.size_of_cdh        = Extra;
   ZIP_CDHE.start_cdh          = where_to;// + SolidN*sizeof(ZIP_CDH
);


   if(fwrite(&ZIP_CDHE, 1, sizeof(ZIP_CDHE), F2) != sizeof(ZIP_CDHE)
) Error(7, "");

   return 0;
}
```

Zipheads

Util

```
/*****************************************************************
***********
 *
          *
 *                    Usefull junks to keep it rolling
          *
 *
          *
 *          From Xtreme 106 (DLL) package by Sabin, Belu (c) 1999
          *
 *
          *
 *****************************************************************
***********/

#pragma pack (1)

#include <stdio.h>
#include <direct.h>
#include <string.h>
#include <stdlib.h>

#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/utime.h> // to modify the file time
#include <windows.h>

#include "crc.h"
#include "util.h"
#include "solid.h"
#include "console.h"
#include "deflate.h"
#include "recovery.h"
#include "zipheads.h"

#include "exports.h"

__declspec( dllexport ) int Xtreme_Archive_(char *name, unsigned in
t *arhfileno);

unsigned int hash_header;  // hash index of string to be inserted
struct TableData configuration_table[9] =
   {
      //WSIZE_INDEX  MIN_MATCH  MAX_MATCH HASH_BITS  TOO_DISTANT  D
```

Util

```
_REST_BITS
        {           1,          4,        258,        15,       1,
  7,      "32K" },  // no more
        {           2,          4,      10000,        16,       1,
  8,      "64K" },
        {           4,          4,      20000,        17,       1,
  9,      "128K" },
        {           8,          4,      30000,        18,       1,
 10,      "256K" },
        {          16,          4,      40000,        19,       1,
 11,      "512K" },  // it seems it does'nt give much compression
        {          32,          4,      60000,        19,       1,
 12,     "1024K" },
        {          64,          4,      62000,        20,       1,
 13,     "2048K" },
        {         128,          4,      64000,        20,       1,
 14,     "4096K" },
        {         256,          4,      65200,        20,       1,
 15,     "8192K" }
    };

bool            write_melt = 1;
unsigned long   header_pos = 0; // this is not the safer place for a
 initialization !
char            tmp[_MAX_PATH], tmp2[_MAX_PATH];
struct          ARH_Header ARH_H;
struct          XTREME_Header XTREME_H;
unsigned int    Totalout, outcnt;
extern          SolidFileType  *SolidArray;

int             EndOfFile,nice_match;
unsigned int    prev_length;
unsigned int    string_starting;
unsigned int    afis_start;
unsigned int    count_start;
unsigned int    match_start;
unsigned int    in_advance_reading;
unsigned short  max_chain_length;
unsigned short  max_lazy_match;
unsigned short  good_match;
unsigned int    TOO_DISTANT, WSIZE, window_size, HASH_BITS, HASH_SIZ
E, HASH_MASK, WMASK, MAX_DIST, H_SHIFT, MIN_MATCH, MAX_MATCH, L_RES
T_BITS, MIN_LOOKAHEAD;
unsigned char   HASH_MAX = 5;
```

Util

```
#define ErrorMess(a, da, ca, b, db, cb, c, dc, cc, d, dd, cd)  if(M
yXtrStructure.ErrorMessageDisplayFunction)  MyXtrStructure.ErrorMes
sageDisplayFunction(a, da, ca, b,  db, cb, c,  dc, cc, d, da, ca);

void Error(short errorindex, char *additionaltext)
{
  switch(errorindex)
   {
     case  0 : ErrorMess(" #ERROR  1 ", " ", 12, " The file ", "'",
 2, short_str(additionaltext, 60),  "'", 2,
                                          " could not be opened !\n
File does not exists, could not be created, or too many open files"
, " ?!", 2);
               break;

     case  1 : ErrorMess(" #ERROR  2 ", " ", 12, "  Archive ", "'",
 2, short_str(Destination, 60),  "'", 2, " cannot be recognized !!\
n ", " ?!", 2);
                        //TryToRepair();
               break;

     case  2 : ErrorMess(" #ERROR  3 ", " ", 12, "  Archive ", "'",
 2, short_str(Destination, 60),  "'", 2, " is damaged !!\n ", " ?!"
, 2);
                //TryToRepair();
               break;

     case  3 : ErrorMess(" #ERROR  4 ", " ", 12, " Inconsistent dat
a crc, data write failed ", "", 2, "",  "", 2, " ", " ?!", 2);
                        break;


     case  4 : ErrorMess(" #ERROR  5 ", " ", 12, "Inconsistent data
 crc, data read failed !", "", 2, "",  "", 2, " ", " ?!", 2);
                        //TryToRepair();
               break;

     case  5 : ErrorMess(" #ERROR  6 ", " ", 12, " Error packing fi
le names ! ", "", 2, "",  "", 2, " ", " ?!", 2);
                        break;

     case  6 : ErrorMess(" #ERROR  7 ", " ", 12, " Error melting fi
le names ! ", "", 2, "",  "", 2, " ", " ?!", 2);
               //TryToRepair();
               break;
```

Util

```
    case  7 : ErrorMess(" #ERROR  8 ", " ", 12, " Error writting t
he archive header ! ", "", 2, "",  "", 2, " ", " ?!", 2);
              break;

    case  8 : ErrorMess(" #ERROR  9 ", " ", 12, " Error reading th
e next header ! ", "", 2, "",  "", 2, " ", " ?!", 2);
              //TryToRepair();
              break;

    case  9 : ErrorMess(" #ERROR 10 ", " ", 12, " Unexpected END-O
F-ARCHIVE encountered ! ", "", 2, "",  "", 2, " ", " ?!", 2);
              //TryToRepair();
              break;

    case 10 : ErrorMess(" #ERROR 11 ", " File size ", 12, Destinat
ion, "size changed since scanning !", 2, " Decompression will no lo
nger be error-free !",  "", 2, " ", " ?!", 2);
              break;

    case 11 : ErrorMess(" #ERROR 12 ", " Archive ", 12, Destinatio
n, " cannot be processed! ", 2, "",  "", 2, " ", " ?!", 2);
              break;

        case 12 : ErrorMess(" #ERROR 13 ", "", 12, Destination, "
seems to have no files ! ", 2, "",  "", 2, " ", " ?!", 2);
              break;

        case 13 : ErrorMess(" #ERROR 14 ", " Unable to modify LOCK
ED archive ", 12, Destination, " Create a new archive by extracting
/deleting/repacking files.", 2, "",  "", 2, " ", " ?!", 2);
              break;

    case 14 : ErrorMess(" #ERROR 15 ", "Recovery record not found
!", 12, Destination, "The archive must have been protected first ('
p'rotect command) !", 2, "",  "", 2, " ", " ?!", 2);
              break;

    case 15 : ErrorMess(" #ERROR 15 ", "Recovery record found !",
12, Destination, "Archive is already protected !", 2, "",  "", 2, "
 ", " ?!", 2);
              break;

        case 16 : ErrorMess(" #ERROR 15 ", "Inconsistent header in
formation !  If header CRC error ", 12, "", " was not reported, DLL
 received garbage data", 2, "",  "", 2, " ", " ?!", 2);
              break;
```

```
  }
}

void Print_Start()
{
  afis_start  = 0;
  count_start = 0x001;
}

int CreateTemporaryArchive(char *DestinationDir, char *temp_name)
{
 char  tempex[ _MAX_PATH ];
 memset(tempex, 0, sizeof(tempex));
 memset(temp_name, 0, _MAX_PATH);
 strcpy(tempex, DestinationDir);
 strcat(tempex, "xtremly.tmp");
 if((F2 = fopen(tempex, "wb+"))==NULL) return 1;
 strcpy(temp_name, tempex);
 return 0;
}

int file_exists(char *disk, char *filename)
{ char tmp[_MAX_PATH];

  memset(tmp, 0, sizeof(_MAX_PATH));
  if(disk[0])
  {
    strcpy(tmp, disk);
    if(tmp[strlen(tmp) -1] != '\\') strcat(tmp, "\\");
  }
  strcat(tmp, filename);
  return (access(tmp, 0) == 0);
}

short IntoarceProcent(unsigned long A, unsigned long B)
 {  unsigned long Oli, Oli1;

    if(A > 1000000L) Oli = A/1000, Oli1 = B/1000;
      else Oli = A, Oli1 = B;
    return (short)((Oli * 1000) / (Oli1 == 0 ? 1 : Oli1));
 }

char *short_str(char *olds, int cat)
{
  static char tmp3[_MAX_PATH];
```

Util

```
  if(strlen(olds) > (unsigned int)cat)
   {
     memset(tmp3, 0, sizeof(tmp3));
     strcpy(tmp3, "..");
     strcat(tmp3, (char *)&olds[strlen(olds) - cat + 2]);
     return tmp3;
   }
  else
    return olds;
}


extern void window_fill_in();

void initialize_engine(short  pack_level)
{
    register unsigned int  j;

    if(Fereastra == NULL || prev == NULL || head == NULL) { Error(1
1, ""); exit(0); }
    for (j = 0;  j < HASH_SIZE; j++)
    {
        head[j].Li = (unsigned short)0x0000;
        head[j].Hc = (unsigned char)0x00;
    }
    string_starting  = 0;
    max_lazy_match = 32; good_match = 8; nice_match = 128; max_chai
n_length = 1024;
    in_advance_reading =  read_from_file((unsigned char *)Fereastra
, (unsigned int)WSIZE);
    if (in_advance_reading == 0 || in_advance_reading == (unsigned
int  )EOF)   { EndOfFile = 1, in_advance_reading = 0; return; }
    EndOfFile = 0;
    while (in_advance_reading < MIN_LOOKAHEAD && !EndOfFile) window
_fill_in();
    hash_header = 0;
}

void Split(char *Name, char *drv, char *dir, char *fnam, char *ext)
{
  _splitpath(Name, drv, dir, fnam, ext);
  //_strlwr(drv != NULL ? drv : " ");   _strlwr(dir != NULL ? dir :
 " ");
  //_strlwr(fnam != NULL ? fnam : " ");  _strlwr(ext != NULL ? ext
: " ");
}
```

Util

```
void ComplexSplit(char *Name, char *drv, char *dir, char *fnam, cha
r *ext)
{
  FILE *U;
  char curr[_MAX_PATH];

  U = fopen(Name, "r");
 _splitpath(Name, drv, dir, fnam, ext);

  if(U != NULL)  fclose(U);
  else
  {
    // a directory or a directory withh a *.c, *., *.????
    _getcwd(curr, sizeof(curr));
    if(_chdir(Name) == -1)
    {
      // must have a special *.*, or *. because we cannot change th
e path rightly
    }
    else
    {
     _chdir(curr); // otherwise the updated archive will come here
!
     if(dir  != NULL)
     {
        strcat(dir, fnam);
                if(ext[1] != '*') strcat(dir, ext);
                CheckSlash(dir);  //  if(dir[strlen(dir) -1] != '\\
') strcat(dir, "\\");
        memset(fnam, 0, sizeof(fnam));
     }
    }
     if(fnam != NULL && fnam[0] == 0) strcpy(fnam,  "*");
     if(ext  != NULL && ext [0] == 0) strcpy(ext,  ".*");
     if(dir [0] == '\\') { strcpy(dir, (char *)&dir[1]); strcat(drv
, "\\"); }
  }
 //_strlwr(drv != NULL ? drv : " ");   _strlwr(dir != NULL ? dir :
" ");
 //_strlwr(fnam != NULL ? fnam : " ");  _strlwr(ext != NULL ? ext :
 " ");
}

void GetCurDir(char *temp)
{
```

Util

```
  memset(temp, 0, _MAX_PATH);
 _getcwd(temp, _MAX_PATH);
  if(temp[strlen(temp) - 1] != '\\') strcat(temp, "\\");
}

void GetTemporaryDirectory(char *buffer)
{
  memset(buffer, 0, _MAX_PATH);
  GetTempPath(_MAX_PATH, buffer);
  if(buffer[strlen(buffer) - 1] != '\\') strcat(buffer, "\\");
}


int _unlink_path(char *path, char kill_directory)
{
   char drive  [_MAX_PATH];
   char direct [_MAX_PATH];
   char dir    [_MAX_DIR];
   char fname  [_MAX_FNAME];
   char ext    [_MAX_EXT];
   int  er;
  memset(direct, 0, sizeof(direct)); memset(drive, 0, sizeof(drive
));  memset(dir,   0, sizeof(dir)); memset(fname, 0, sizeof(fname))
;  memset(ext,   0, sizeof(ext));

   _splitpath(path, drive, dir, fname, ext);
   strcat(drive, dir);
   strcat(direct, drive);
   strcat(drive, fname);
   strcat(drive, ext);
   er = _unlink(drive);
   if(kill_directory) _rmdir(direct);
   return er;
}

void MkDir(char *MDir)
 {  unsigned char a=0, j=0;
    char          TM[_MAX_PATH];

    // This is my special MKKDIR procedure :
    // Creates a special directory like
    //   "C:\\D\\T\\Y\\u7\\7\12\\12121\\6565\\87878\\787
    // required when decompression, even if for instance D\\T\\Y\\U
7
    // do not exist!
```

```
    memset(TM, 0, _MAX_PATH);
    while ((a = MDir[j++]) != 0)
     {
       if(a == '\\')   mkdir(TM);
       TM[strlen(TM)] = a;
     }
 }

int Verify(char *Name, char *Symbol, char LenN, char LenS) // Symbo
l=????top, ==> Name=desktop
{ char j;
  char m =  LenS < LenN ? LenS : LenN;
  char S[300];
  char N[300];

  if((!strpbrk(Symbol,"*")) && (LenS != LenN)) return 0;

  strcpy(S, Symbol);  strlwr(S);
  strcpy(N, Name  );  strlwr(N);
  for (j=0; j < m; j++)
   {
     if(S[j] == '*' || S[j]=='!' || S[j]=='?'); // Can be anything
       else
     if(S[j] != N[j]) return 0;
  }
  return 1;
}

extern XTREME106_CMDSTRUCTURE  MyXtrStructure;

void Fclose(FILE *O, unsigned int _SolidArrayIndex, unsigned int al
lowed)
{
   struct _utimbuf times; // set file's modification time !
   char RealPathToFile[_MAX_PATH*2];

   memset((unsigned char *)RealPathToFile, 0, sizeof(RealPathToFile
));
   if(O)
   {
         fclose(O);
         if(allowed)
         {
       times.actime  =  (time_t)SolidArray[_SolidArrayIndex].time_d
ate; //(time_t)(date_and_time);
       times.modtime =  (time_t)SolidArray[_SolidArrayIndex].time_d
```

```
ate; //(time_t)(date_and_time);

            strcpy(RealPathToFile, MyXtrStructure.extractionDirector
y);
            CheckSlash(RealPathToFile);
          strcat(RealPathToFile ,SolidArray[_SolidArrayIndex].RealName
);

         SetFileAttributes(RealPathToFile/*SolidArray[_SolidArrayInde
x].RealName*/, 32);

            // first, we clear the attribute, then set the right one
!

            _utime(RealPathToFile/*SolidArray[_SolidArrayIndex].Real
Name*/, &times);
            SetFileAttributes(RealPathToFile/*SolidArray[_SolidArray
Index].RealName*/, SolidArray[_SolidArrayIndex].atrib);
          _utime(RealPathToFile/*SolidArray[_SolidArrayIndex].RealName
*/, &times);

          }
   }
}

int OpenFile(char *Nam, unsigned int date_and_time, unsigned char a
ttrib)
{  char dir[_MAX_PATH];
   char dir2[_MAX_PATH];
   char  Tmp[_MAX_PATH];
   char Name[_MAX_PATH];

   memset(dir2, 0, _MAX_PATH); memset(dir, 0, _MAX_DIR); memset(Tmp
, 0, _MAX_PATH); memset(Name, 0, _MAX_PATH);
   if(extractionDestination[0])
   {
     strcpy(Name, extractionDestination);
     //if(Name[strlen(Name) - 1] != '\\') strcat(Name, "\\");
     CheckSlash(Name);
   }
   else
   {
     getcwd(Name, _MAX_PATH);
     //if(Name[strlen(Name) - 1] != '\\') strcat(Name, "\\");
     CheckSlash(Name);
   }
```

```
    strcat(Name, Nam);

    Split (Name, dir, dir2, NULL, NULL);
    strcat(dir,  dir2);
    strcpy(Tmp,  dir);
    MkDir(dir);

    if(AssumeYes == 0)
    {
      if((F2  = fopen(Name, "rb" )) != NULL )
      {
        fclose(F2);
        sprintf(tmp, " Overwrite %s ? (Y)es,(E)nter/(N)o/(A)ll/(C)an
cel", Name);
            Printf(tmp, "", 11);

        char t = ScanForAchar(tmp, 4);
            Printf("", "\n", 11);
        switch(t)
          {
             case 'A' : case 'a' : write_melt = 1; AssumeYes = 1; br
eak;
             case 'C' : case 'c' : write_melt = 0; exit(1);       br
eak;
             case 'Y' : case 'y' :
             case 'E' : case 'e' :
                        case 13  : write_melt = 1;                br
eak;
             case 'N' : case 'n' : write_melt = 0;                re
turn 0;
          }
      }
    }

  // since the file cannot be overwritten if it has a read only atr
ibute,
  // we must take care and overwrite it anyway, why bother and ask
the user ?

  SetFileAttributes(Name, 32);  // okay, it works okay
  if((F2  = fopen(Name, "wb+" )) == NULL )  Error(0, Name);
  return 1; // errno == 0
}

int Check_ARH_Header(struct ARH_Header *Ar)
```

Util

```
{
  unsigned int ciroco;

  // must be rewritten, too unsafe ... header may contain garbage d
ata ...
  if((unsigned char)Ar->_HeaderCRC  != 0xFA) return 0;
  local_uint_crc((unsigned char *)Ar,  (unsigned int) (sizeof(struc
t ARH_Header) - 4 /* without the CRC itself */), (unsigned int *)&c
iroco);
  if((unsigned  int)Ar->_UHeaderCRC != (unsigned int)  ciroco) retu
rn 0;

  return 1; // okay !
}

int FWrite_ARH_Header(struct ARH_Header *Ar, FILE *F)
{
  // we have a header, filled with something ....
  // just add the CRC, and it's on the file!
  unsigned int ciroco;
  unsigned char header_map[_MAX_PATH * 2];
  unsigned int  header_map_len = (unsigned int)(sizeof(struct ARH_H
eader) - 4); /* without the CRC itself */

  Ar->_HeaderCRC  = 0xFA; // fill it before it gets random

  memset((unsigned char *)&header_map, 0, sizeof(header_map));
  memcpy((unsigned char *)&header_map, (unsigned char *)Ar, header_
map_len);

  local_uint_crc((unsigned char *)&header_map, header_map_len, (uns
igned int *)&ciroco);
  Ar->_UHeaderCRC = (unsigned int)ciroco;

  if(Ar->_recovery_Compressed)
  {
     // there must be an error, cause I did not initialized a shit
on these !
        //int u = 0;
        //printf(" \n\n\n error --> damaged file header detected .
..\r\n");
        //Printf(" \n\n\n error --> damaged file header detected .
..\r",  "\n\n ", 9);
        //Sleep(4444);
     //u++;
  }
```

Util

```
  if(fwrite(Ar, 1, sizeof(struct ARH_Header), F) != sizeof(struct A
RH_Header)) return 1;
  else return 0;
}

int  Write_Archive_Header(void)
{
  memset((unsigned char *)&(ARH_H._Name), 0, sizeof(ARH_H._Name));
  strcpy(ARH_H._Name,    "ULEB106");

  //if(Actualizez) ARH_H._SolidN += SolidN;  // please re-check!
  //else
  ARH_H._SolidN = SolidN; // I think this holds even when updating
...

  ARH_H._Solida = Solida;
  ARH_H._Version = 106;
  ARH_H._Locked = 0x00;
  ARH_H._SolidaTableSize = TableIndex;

  if(ZipCloaking == 0)
  {
    // Okay, befor writting the Header down, we have to fill in the
 CRC !
    // So, we will replace fwrite cu FWrite_ARH_Header
        //if(fwrite(&ARH_H, 1, sizeof(ARH_H), F2) != sizeof(ARH_H))
  Error(7, "");

        if(FWrite_ARH_Header(&ARH_H, F2)) Error(7, "");
  }
  else
  {
    if(Solida == 0) { Printf(" -s [solid mode] required on the comm
and line", " ", 4); exit(0);}
    WriteFirstZipHeader();
  }
 return 0;
}

int FileSelection(void)
{
  char TmpDirek1[_MAX_PATH], TmpDirek2[_MAX_PATH], UpdateExitanceFi
leName[_MAX_PATH];
  unsigned int i, j;
```

Util

```
cted == 1))  write_melt = 0;
  // Chestia de deasupra e ca nu decomprimam fisierele pe care le s
tergem, la DELETE/UPDATE
  // deci exact ce ne trebuia, adica, punem un || Extrag == 1 && ..
... pe care le vom completa
  // noi ..... Deci trebuie selectate toate fisierele care apartin
de Sursele ...
  // de exemplu : "program files\"..... all here are selected
  // if the ""program files\" is in the sursele
  // the single problem is that the sursele must contain the Archiv
eFullPath, and if
  // directory to end with a '\\'


  // I don't think I can rely on SolidN but it is setted when enter
ing the archive ...
  // will have to check if the questioned file is
  // among the MyXtrStructure.Sursele or not ...
  // nu among the Solid Files, ci among Sursele, and the maximum is
 MaxDirectories ...

  for(i=0; i < SolidN; i++)  SolidArray[i].selected = (Sterg || Act
ualizez ? 0 : 1); // this will not be decompressed (marked to be de
leted from queue)

  for(j=0; j < MyXtrStructure.MaxDirectories; j++)
  {
    // okay, so extract a Sursele, add the Path1 or 2 which is now
the archivePath ...
        // and so on ...

    memset(TmpDirek1, 0, sizeof(TmpDirek1));
    strcpy(TmpDirek1, (MyXtrStructure.Sursele[j].Path1Or2 ? MyXtrSt
ructure.Path1 : MyXtrStructure.Path2));
        CheckSlash(TmpDirek1);
    strcat(TmpDirek1, MyXtrStructure.Sursele[j].Name);
        strlwr(TmpDirek1);

        // update patch only .....
        if(strstr(TmpDirek1, ":\\") != NULL)
        {
       // the file path is with "DISK:\\", when updating, so this p
rocedure needs patches, but it is the best ...
          memset(UpdateExitanceFileName, 0, sizeof(UpdateExitanceFi
leName)); //the name with full path
      strcpy(UpdateExitanceFileName, TmpDirek1);
```

Util

```
            memset(TmpDirek2, 0,  sizeof(TmpDirek2));
        strcpy(TmpDirek2,     (char *)&TmpDirek1[3]);
            memset(TmpDirek1, 0,  sizeof(TmpDirek1));
        strcpy(TmpDirek1,     TmpDirek2);

            memset(TmpDirek2, 0, sizeof(TmpDirek2));
          }

    if(MyXtrStructure.Sursele[j].File == 0) strcat(TmpDirek1, "\\")
; // e directory, add '\\'

    for(i=0; i < SolidN; i++)
          {
      memset(TmpDirek2, 0, sizeof(TmpDirek2));
      strcpy(TmpDirek2, SolidArray[i].RealName);
          strlwr(TmpDirek2);

          if(strstr(TmpDirek2, TmpDirek1) == TmpDirek2)
          {
                // this file must be selected .... for writing ...
                // basically it's the other way around, but now we
are not deleting the file
                // but selecting the others to be deleted
        // Bine bine, dar nu fac update pe un fisier, doar ca direc
torul a fost selectat, ci il sterg doar
                // daca el exista cu adevarat, si va fi replaced ..
..
        if(Actualizez == 0)
                {
                  SolidArray[i].selected = (Sterg || Actualizez ? 1
 : 0); // this will be decompressed, but I want it deleted now
                }
                else
                {
          if(_access(UpdateExitanceFileName, 0 ) == 0) //*real path
 here*/
                        SolidArray[i].selected = (Sterg || Actuali
zez ? 1 : 0); // so only if exists ... it will be replaced, even if
 the directory is here ...
                }

              if(MyXtrStructure.Sursele[j].File) goto NextFile; /
/ after the file is found, no time to waste for another file ... ah
 ?
```

Util

```
        }
    NextFile : ; // just a j++
  }
  return 0;
}

// COMPRESSION USE ONLY

int read_from_file(unsigned char *buf, unsigned int siz)
{   unsigned int len, /*ratio,*/ x;
    long size = siz, verif;

    len = fread((unsigned char *)buf, 1, size, F1);

    if ((len == (unsigned)(-1) || len == 0) && Solida)
      {
        len = TotalRead = 0;
        for(;;)
        {
          Fclose(F1, 0, 0);

                  //
                  //  Gata aici stergem fisierul
                  //
                  //


                  if(MyXtrStructure.sfx && SolidK)
                  {
                     char RealPathToFile[_MAX_PATH*2];
              memset((unsigned char *)RealPathToFile, 0, sizeof(Real
PathToFile));
                     strcpy(RealPathToFile ,SolidArray[SolidK-1].Re
alName);
                        _unlink(RealPathToFile);
                  }

                  if(SolidN == SolidK)
                  {
                          if(MyXtrStructure.sfx)
                          {
                         _rmdir(MyXtrStructure.extractionDirectory);
// cleaning up everything :)
                          }
                     break;
                  }
```

Util

```
            sprintf(tmp, "%-69s", short_str(InFile, 69));
            Print(Repack ? " Repacked " : (ZipCloaking ? " Zipped " :
 " Freezed "), "", ZipCloaking ? 14 : 3, tmp, "\n", 2, "", "", 0, "
", "", 0, -1);

            memset(InFile, 0, _MAX_PATH);
            strcpy(InFile, SolidArray[SolidK].RealName);
            ratio = IntoarceProcent(count_start + len, TotalSolidSize
);
            sprintf(tmp, "%-60s", short_str(InFile, 60));
            sprintf(tmp2," %3d.%d%c", ratio / 10, ratio % 10, '%');

                  Print(Repack ? " Repacking " : (ZipCloaking ? " Z
ipping " : " Freezing "), "", ZipCloaking ? 14 : 3, tmp, " ", 2, t
mp2, "\r", 3, "", "", 0, (ratio/10));

            TotalSum = SolidArray[SolidK].size;
            if((F1 = fopen(InFile, "rb" )) == NULL ) Error(0, InFile)
;
            verif = _filelength(fileno(F1));
            if(verif != TotalSum) Error(10, "");     // file size chan
ged since scanning !!!!
            x = fread((unsigned char *)buf + len, 1, size, F1);
            len += x;
            SolidK++;
            if((size -= x) <= 0) break;
         }
      }
   TotalRead += len;
   return (int)len;
}

 // DECOMPRESSION Use Only

void flush_window(unsigned int Cati)
{
 // We must avoid the extraction of files which will eventually be
 // deleted, so will put a simple falg to simulate this !
 // Remember that if a file is selected to be erased -->
 // SolidArray[i].selected = 1, and write_melt == 0;

 if(Cati == 0) return;
 Solid_CRC = INIT_CRC_VALUE;
 updcrc((unsigned char *)Fereastra, (unsigned int)Cati, &Solid_CRC)
```

Util

```
TotalSum -=Cati;

if(Solida && (TotalSum < 0))  // we have to split the file
 {
  TotalSum += Cati;
  if(write_melt)
    {
     fwrite((unsigned char *)Fereastra, 1, (unsigned int)TotalSum,
F2);
     sprintf(tmp, " %-64s" , short_str(InFile, 64));
     if(Repack == 0)
         {
            Print(tmp, "", 2, " melted", "", 3, "", "", 0, "", "",
0, -1);
         }
     Fclose(F2, SolidK ? SolidK - 1 : SolidK, 1);
   }
   unsigned long Sum = 0, FileS = 0;
   for(;;)
   {
     if(SolidK >= SolidN) break;
     memset(InFile, 0, _MAX_PATH);
     strcpy(InFile, SolidArray[SolidK].RealName);
     FileS = SolidArray[SolidK].size; // first file's size

     if((Sterg == 1 || Actualizez == 1 || Extrag == 1) && (SolidAr
ray[SolidK].selected == 1))
                  write_melt = 0;
     else
                   write_melt = 1;
     if(write_melt)  OpenFile(InFile, SolidArray[SolidK].time_date
 SolidArray[SolidK].atrib);

     SolidK++;
     if(FileS > Cati - TotalSum - Sum)
     {
       if(write_melt) fwrite((unsigned char *)Fereastra + TotalSum
+ Sum, 1, Cati - TotalSum - Sum, F2);
       TotalSum = FileS - (Cati - TotalSum - Sum);
       break;
     }
     else
     {
       if(write_melt)
       {
         fwrite((unsigned char *)Fereastra + TotalSum + Sum, 1, Fi
```

Util

```
leS, F2);
          sprintf(tmp, " %-64s" , short_str(InFile, 64));
          if(Repack == 0)
                  {
                        Print(tmp, "", 2, " melted", "", 3, "", ""
, 0, "", "", 0, -1);
                  }
         }
        Fclose(F2, SolidK ? SolidK - 1 : SolidK, 1);
       }
      Sum += FileS;
     }
    }
   else
    {
      if(write_melt)
      {
        short ratio = IntoarceProcent(TotalSum, (Solida ? TotalSoli
dSize : TotalRead));
        fwrite((unsigned char *)Fereastra, 1, Cati, F2);
        sprintf(tmp,  "%-63s", InFile);
        sprintf(tmp2, "%2d.%d%c", ratio/10, ratio%10, '%');
        if(Repack == 0) Print(" Melting ", " ", 3, tmp, " ", 3, tmp
2, "\n", 2, "" , "",  0, -1);
      }
  }
}


void SkipToNext_ZIP(struct ZIP_Header *Head, FILE *U)
{
  unsigned int x = Head->extra_len;
  while(x--) getc(U);
  fseek(U, Head->compressed, SEEK_CUR);
}

void GotoTheSolidPackedFiles(FILE *U)
{ char tmpFil[_MAX_PATH];

  rewind(F1);
  if((fread(&ZIP_H, 1, sizeof(ZIP_H), F1)) !=  sizeof(ZIP_H))  Erro
r(1, "");
  if((fread(&tmpFil, 1, ZIP_H.filenamel, F1)) !=  ZIP_H.filenamel)
 Error(1, "");
```

Util

```
    while(x--) getc(U);
  }

  int Read_ZIP_ARH_Files()
  {
    char tmpFil[_MAX_PATH];

    rewind(F1);
    SolidN = TotalSolidSize = 0;
    for(;;)
    {
      UpdateSolidArray(SolidN);
      memset(tmpFil, 0, sizeof(tmpFil));

      if((fread(&ZIP_H, 1, sizeof(ZIP_H), F1)) !=  sizeof(ZIP_H))  Er
 ror(1, "");
      if(ZIP_H.signature != 0x04034b50)  break;
      if((fread(&tmpFil, 1, ZIP_H.filenamel, F1)) !=  ZIP_H.filenamel
 )  Error(1, "");

      SkipToNext_ZIP(&ZIP_H, F1);
      TotalSolidSize += ZIP_H.uncompressed;
      SolidFillTheName(SolidN, tmpFil, ZIP_H.uncompressed, 0, 0);  //
 check for the time = 0
      SolidN++;
    }
  GotoTheSolidPackedFiles(F1);
  return 0;
  }

  int Xtreme_Archive_(char *name, unsigned int *arhfileno)
  {
    int x = 0;
    FILE  *U;

    if((U = fopen(name, "rb")) == NULL) x = 0;
    else
    {
      if((fread(&ARH_H, 1, sizeof(ARH_H), U)) !=  sizeof(ARH_H))  x
 = 0;
          if((x = Check_ARH_Header(&ARH_H)) == 1)
                  *arhfileno = ARH_H._SolidN;
          else
                  *arhfileno = 0;
    }
    Fclose(U, 0, 0);
```

Util

```
  return x;
}

int Read_Archive_Header(char initialize_tables)
 {
    Read_ZIP_ARH_header();
    Solida = ARH_H._Solida;
    if(Solida && initialize_tables)
     {
       TableIndex = ARH_H._SolidaTableSize;
       ReadjustTables();
     }
    return 0;
 }

void SkipToNextFile(FILE *FF)
{
     rewind(FF);
     fseek(FF, header_pos + sizeof(ARH_H), SEEK_SET);
}

int CheckStruct(char *P)
 {
    long u = ftell(F1), o;
    o = fread(&XTREME_H, 1, sizeof(XTREME_H), F1);

    if((o != sizeof(XTREME_H)) && (o == 0)) return 1; // nothing to
 read !
    if((o != sizeof(XTREME_H)) && (o))       Error(9,"");   // unabl
e to read !

    TotalRead    = TotalSum = XTREME_H._SizeUnCom;
    TotalWrit    = XTREME_H._NextFile - sizeof(XTREME_H) - XTREME_H.
_NamLen;
    header_pos += XTREME_H._NextFile;// + i0; // huffman tables, wh
ich can actually be more and more
    if((fread(P, 1, XTREME_H._NamLen,  F1)) != XTREME_H._NamLen) Er
ror(9,"");

    TableIndex = (char)XTREME_H._Table;
    if(Listez == 0 && Repack == 0) ReadjustTables();

    if(TotalWrit<0 || P[0]==0)
     {
      if(ARH_H._recovery_Filesize == 0)  Error(8, ""); // if we are
not reading the recovery records !
```

Util

```
      return 1;
     }
    return 0;
}

void Read_ZIP_ARH_header()
{ unsigned int u;
  char TmpFil[_MAX_PATH];

  rewind(F1);
  u = getw(F1);

  if(u == 0x04034b50) // PK
  {
    ZipCloaking = 1;
    rewind(F1);
    if((fread(&ZIP_H, 1, sizeof(ZIP_H), F1)) !=  sizeof(ZIP_H))  Er
ror(1, "");
    if((fread(&TmpFil, 1, ZIP_H.filenamel, F1)) !=  ZIP_H.filenamel
)  Error(1, "");
    if((fread(&ARH_H, 1, sizeof(ARH_H), F1)) !=  sizeof(ARH_H))  Er
ror(1, "");
  }
  else
  if(u == 0x42454C55) // ULEB
  {
     ZipCloaking = 0;
     rewind(F1);
     if((fread(&ARH_H, 1, sizeof(ARH_H), F1)) !=  sizeof(ARH_H))  E
rror(1, "");

        // now let's check to see that it's quite an Xtreme Archiv
e
        // I mean Compute CRC and compare!

        if(Check_ARH_Header(&ARH_H) == 0) Error(1, ""); // does no
t match!
  }
  else Error(1, "");
}

void GetArchiveType(char *archive_type)
{
  if(F1 == NULL) *archive_type = -1;
```

Util

```
   *archive_type = ARH_H._Solida;
   if(Solida) TableIndex = ARH_H._SolidaTableSize;
   rewind(F1);
 }

 void DamageProtectorArchive(void)
 {
   unsigned int U = _filelength(fileno(F1));

   InitializeProtection(metoda, U, 1);
   InitializeCorrectionForFile(F1, U, 1, metoda);
   ComputeDataRecoveryRecord(F1);
 }

 void RepairArchive(void)
 {
   COMPRESS_RECOVERY_RECORD = ARH_H._recovery_Compressed;
          // essential, oh my God!
   InitializeProtection(ARH_H._recovery_Method, ARH_H._recovery_File
size, 0);
   InitializeCorrectionForFile(F1, ARH_H._recovery_Filesize, 0, meto
da);

   Repair();

   Fclose(F1, 0, 0);
   Fclose(F2, 0, 0);
 }

 void TestRecoveryRecord(void)
 {
   if(ARH_H._recovery_Filesize)
   {
     // outperform a recovery record testing based upon each block'
s crc !
     COMPRESS_RECOVERY_RECORD = ARH_H._recovery_Compressed;
        // essential, oh my God!
        if(COMPRESS_RECOVERY_RECORD > 1)
        {
              Error(16, " DLL garbage DATA detected .....");
             // return;
             // this is only for the error to be spotted !!!
        }
    InitializeProtection(ARH_H._recovery_Method, ARH_H._recovery_Fi
lesize, 2);
```

Util

```
    InitializeCorrectionForFile(F1, ARH_H._recovery_Filesize, 2, me
toda);
    CheckDataRecovery(F1);
  }
}
```

Recovery

```
/*****************************************************************************
 *
          *
 *         Error Correcting Codes (ECC) adding/removing procedures
          *
 *
          *
 *                    Protection at it's best, belive me
          *
 *
          *
 *           From Xtreme 106 (DLL) package by Sabin, Belu (c) 1999
          *
 *
          *
 *****************************************************************************/

/*
=============================================================================
=======================
                          7.C DAMAGE PROTECTION TECHNOLOGY
=============================================================================
=======================

     #define NON_CONSECUTIVE_NO                6
     #define PROTECTED_BUFFER_NO              10
     #define PROTECTED_BUFFER_SIZE          8192

     [P] [NON_CONSECUTIVE_NO]
     [R]   0  1  2  3  4  5
     [O]   6  7  8  9 10 11
     [T]  12 13 14 15 16 17  <=== The Sectors are no longer
     [.]  ... X  X  X  X  X       consecutive, but interlaced.
     [.]   X  X  X  X  X  X       So, even if the file is damaged
     [N]   X  X  X  X  X  X       badly, seqvential or not, the
     [O]   X  X  X  X  X  X       chances are all the same to recove
r
          ---------------------
 XOR_Rez  [0] [1] [2] [3] [4] [5]

*/

#pragma pack (1)
```

Recovery

```
#include <io.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "crc.h"
#include "huffman.h"
#include "util.h"
#include "console.h"
#include "recovery.h"
#include "zipheads.h"

int DATA_RECOVERY_INDEX;  // sizeof(ARH_H) //6 -> that extra header

char    *A[8] = {"-\0", "\\\0", "|\0", "/\0", "-\0", "\\\0", "|\0",
 "/\0"};
char     Tmp__[ _MAX_PATH];

unsigned short    VAR_NON_CONSECUTIVE_NO =   3;
unsigned short   VAR_PROTECTED_BUFFER_NO =   5;
unsigned short VAR_PROTECTED_BUFFER_SIZE = 512;


unsigned char Table[PROTECTED_BUFFER_SIZE], metoda = 0, COMPRESS_RE
COVERY_RECORD = 0;
unsigned char XOR_Rez[NON_CONSECUTIVE_NO][PROTECTED_BUFFER_SIZE];
unsigned char Final[NON_CONSECUTIVE_NO*PROTECTED_BUFFER_NO*PROTECTE
D_BUFFER_SIZE + 3];

struct          ReadStructure ReadIn[NON_CONSECUTIVE_NO];
struct          Protect Protector;

long            fftrunc;
unsigned int    jok=0, SectorsNo, SectorsBadNo, SectorsRepairedNo, FL
en, SectorsK,
                Sectoros, WhereData = 0, WhereFile/* = DATA_RECOVERY_
INDEX*/, ProtectorSize = sizeof(Protector);

void InitializeCorrectionForFile(FILE *FF, unsigned int size, char
protect, unsigned char metode)
{
  if(protect != 2)
      Printf(" Scanning, please wait ... ", " ", 2);
```

Recovery

```
   FLen = size;
   SectorsNo  = (FLen / VAR_PROTECTED_BUFFER_SIZE) + 2;

   Protector.recovery_     = (struct DataRecovery *)malloc(sizeof(str
uct DataRecovery) * (VAR_NON_CONSECUTIVE_NO + 1));
   Protector.SubSectorCrc = (unsigned short *)malloc(sizeof(unsigned
 short) * (VAR_NON_CONSECUTIVE_NO * VAR_PROTECTED_BUFFER_NO + 1));

   if((Protector.recovery_ == NULL) || (Protector.SubSectorCrc == NU
LL))  { Error(11, ""); return; }

   if(protect == 1)
   {
     rewind(F1);
     ARH_H._recovery_Filesize  = size;
     ARH_H._recovery_Method    = metoda;
     ARH_H._recovery_Compressed = COMPRESS_RECOVERY_RECORD;
     if(ZipCloaking) fseek(F1, sizeof(ZIP_H) + ZIP_H.filenamel, SEEK
_SET);

        // if(fwrite(&ARH_H, 1, sizeof(ARH_H), F1) != sizeof(ARH_H)
)  Error(7, "");
        if(FWrite_ARH_Header(&ARH_H, F1)) Error(7, ""); // rewritte
n, has a 4 byte CRC built in!

     fseek(F1, DATA_RECOVERY_INDEX, SEEK_SET);
     WhereData = FLen + DATA_RECOVERY_INDEX;
   }
   else
   {
     fseek(FF, FLen + DATA_RECOVERY_INDEX, SEEK_SET);
     WhereData = FLen + DATA_RECOVERY_INDEX;
     fftrunc = ARH_H._recovery_Filesize;
   }
   if(protect != 2) Printf (" Scanning complete ", "!", 2);
}

int WriteProtectStructure(FILE *FF, char flush)
{
 unsigned int  i, k =0;
 unsigned char buffer1[10240];//VAR_NON_CONSECUTIVE_NO * VAR_PROTEC
TED_BUFFER_NO * 2 + VAR_NON_CONSECUTIVE_NO * sizeof(struct DataReco
very) + sizeof(struct Protect)*2];
 unsigned char buffer2[10240];//VAR_NON_CONSECUTIVE_NO * VAR_PROTEC
TED_BUFFER_NO * 2 + VAR_NON_CONSECUTIVE_NO * sizeof(struct DataReco
very) + sizeof(struct Protect)*2];
```

Recovery

```
  if(COMPRESS_RECOVERY_RECORD)
  {
    memcpy((unsigned char *)&buffer1[ k ], (unsigned char *)&Protect
or.n_subsectors, sizeof(unsigned short));
    k += sizeof(unsigned short);
    memcpy((unsigned char *)&buffer1[ k ], (unsigned char *)&Protect
or.n_buffer_size, sizeof(unsigned int));
    k += sizeof(unsigned int);

    for(i=0; i< VAR_NON_CONSECUTIVE_NO; i++)
    {
      memcpy((unsigned char *)&buffer1[ k ], (unsigned char *)&Prote
ctor.recovery_[i].CRC, sizeof(unsigned short));
      k += sizeof(unsigned short);
      memcpy((unsigned char *)&buffer1[ k ], (unsigned char *)&Prote
ctor.recovery_[i].buffer[0], PROTECTED_BUFFER_SIZE);
      k += PROTECTED_BUFFER_SIZE;
    }

    for(i=0; i< (unsigned)(VAR_NON_CONSECUTIVE_NO * VAR_PROTECTED_BU
FFER_NO); i++)
    {
      memcpy((unsigned char *)&buffer1[ k ], (unsigned char *)&Prote
ctor.SubSectorCrc[i], sizeof(unsigned short));
      k += sizeof(unsigned short);
    }

    EncodeBuffer(k, buffer1, buffer2, &i);
    putw(k, F1);  putw(i, F1);
    if(fwrite(buffer2, 1, i, F1) != i) return 1;
  }
  else
  {
    if((fwrite((unsigned short*)&Protector.n_subsectors,  1, sizeof(
unsigned short), F1)) != sizeof(unsigned short)) return 1;
    if((fwrite((unsigned int  *)&Protector.n_buffer_size, 1, sizeof(
unsigned int  ), F1)) != sizeof(unsigned int  )) return 1;

    for(i=0; i< VAR_NON_CONSECUTIVE_NO; i++)
     if((fwrite((struct DataRecovery *)&Protector.recovery_[i], 1, s
izeof(struct DataRecovery), F1)) != sizeof(struct DataRecovery)) re
turn 1;

    for(i=0; i< (unsigned)(VAR_NON_CONSECUTIVE_NO * VAR_PROTECTED_BU
FFER_NO); i++)
```

Recovery

```
    if((fwrite((unsigned short *)&Protector.SubSectorCrc[i], 1,  s
izeof(unsigned short), F1)) != sizeof(unsigned short)) return 1;
 }
 return 0;
}

int ReadProtectStructure(FILE *FF, char read)
{unsigned int i, k=0;
 unsigned char buffer1[10240];
 unsigned char buffer2[10240];

 if(COMPRESS_RECOVERY_RECORD)
 {
   k = getw(F1); i = getw(F1);
   if(fread(buffer2, 1, i, F1) != i) return i;
   DecodeBuffer(k, buffer2,  buffer1, &i);
   k =0;

   memcpy((unsigned char *)&Protector.n_subsectors,  (unsigned char
*)&buffer1[ k ], sizeof(unsigned short));
   k += sizeof(unsigned short);
   memcpy((unsigned char *)&Protector.n_buffer_size, (unsigned char
*)&buffer1[ k ], sizeof(unsigned int));
   k += sizeof(unsigned int);

   for(i=0; i< VAR_NON_CONSECUTIVE_NO; i++)
   {
     memcpy((unsigned char *)&Protector.recovery_[i].CRC, (unsigned
 char *)&buffer1[ k ], sizeof(unsigned short));
     k += sizeof(unsigned short);
     memcpy((unsigned char *)&Protector.recovery_[i].buffer[0], (un
signed char *)&buffer1[ k ], PROTECTED_BUFFER_SIZE);
     k += PROTECTED_BUFFER_SIZE;
   }
   for(i=0; i< (unsigned)(VAR_NON_CONSECUTIVE_NO * VAR_PROTECTED_BU
FFER_NO); i++)
   {
     memcpy((unsigned char *)&Protector.SubSectorCrc[i], (unsigned
char *)&buffer1[ k ], sizeof(unsigned short));
     k += sizeof(unsigned short);
   }
 }
 else
 {
   if((fread((unsigned short*)&Protector.n_subsectors,  1, sizeof(u
nsigned short), FF)) != sizeof(unsigned short))   return 1;
```

Recovery

```
   if((fread((unsigned int  *)&Protector.n_buffer_size, 1, sizeof(u
nsigned int  ), FF)) != sizeof(unsigned int  ))   return 1;
   for(i=0; i< VAR_NON_CONSECUTIVE_NO; i++)
     if((fread((struct DataRecovery *)&Protector.recovery_[i], 1, s
izeof(struct DataRecovery), FF)) != sizeof(struct DataRecovery))  r
eturn 1;

   for(i=0; i< (unsigned)(VAR_NON_CONSECUTIVE_NO * VAR_PROTECTED_BU
FFER_NO); i++)
     if((fread((unsigned short *)&Protector.SubSectorCrc[i], 1,  si
zeof(unsigned short), FF)) != sizeof(unsigned short))   return 1;
 }

 if(Protector.n_subsectors > VAR_NON_CONSECUTIVE_NO * VAR_PROTECTED
_BUFFER_NO) return 1;
 return 0;
}

void ComputeDataRecoveryRecord(FILE *FF)
{
  unsigned int   i, j, totalsize = 0, size=0, p, n, k = 0;
  unsigned int   buffer_index = 0;
  unsigned short SubSectorCRC;
  unsigned int   Index[NON_CONSECUTIVE_NO];

  memset((unsigned char *)&Index, 0, sizeof(Index));
  memset((unsigned char *)&ReadIn, 0, sizeof(ReadIn));
  SectorsNo=0;

  Printf (" Computing data recovery record, please wait ", "...", 3
);

  for(i=0;;i++)
  {
    if((i % (VAR_PROTECTED_BUFFER_NO * VAR_NON_CONSECUTIVE_NO) == 0
) && i)
    {
      if(SectorsNo % 50 == 0) printf(".");
      for(p=0; p < VAR_NON_CONSECUTIVE_NO; p++)
       {
         memset((unsigned char *)&XOR_Rez[p], 0, VAR_PROTECTED_BUFF
ER_SIZE);
         memcpy((unsigned char *)&XOR_Rez[p], (unsigned char *)&(Re
adIn[p].Date[0].buffer[0]), VAR_PROTECTED_BUFFER_SIZE);
```

Recovery

```
            for(j=0; j < VAR_PROTECTED_BUFFER_SIZE; j++)
                XOR_Rez[p][j] ^= ReadIn[p].Date[k].buffer[j];

          memset((unsigned char *)&(Protector/*[SectorsNo]*/.recover
y_[p].buffer[0]), 0, VAR_PROTECTED_BUFFER_SIZE);
          memcpy((unsigned char *)&(Protector/*[SectorsNo]*/.recover
y_[p].buffer[0]), (unsigned char *)&XOR_Rez[p], VAR_PROTECTED_BUFFE
R_SIZE);
          local_short_crc((unsigned char*)&(Protector/*[SectorsNo]*/
.recovery_[p].buffer[0]), VAR_PROTECTED_BUFFER_SIZE, &SubSectorCRC)
;
          Protector/*[SectorsNo]*/.recovery_[p].CRC = SubSectorCRC;
        }

        for(n=0, k=0; k < VAR_PROTECTED_BUFFER_NO; k++)
          for(p=0; p < VAR_NON_CONSECUTIVE_NO; p++)
            Protector/*[SectorsNo]*/.SubSectorCrc[n++] = ReadIn[p].Da
te[k].CRC;

        Protector/*[SectorsNo]*/.n_subsectors  = (unsigned short)(VAR
_PROTECTED_BUFFER_NO * VAR_NON_CONSECUTIVE_NO);
        Protector/*[SectorsNo]*/.n_buffer_size = size;

        SectorsNo++;
        if(WriteCorrectionTable(FF, 0))  break;

        i = size = buffer_index = 0;
        memset((unsigned char *)&Index, 0, sizeof(Index));
        memset((unsigned char *)&ReadIn, 0, sizeof(ReadIn));
      }

      memset((unsigned char *)&Table, 0, VAR_PROTECTED_BUFFER_SIZE);
      n = fread ((unsigned char *)&Table, 1, totalsize + VAR_PROTECTE
D_BUFFER_SIZE >= FLen - DATA_RECOVERY_INDEX ? VAR_PROTECTED_BUFFER_
SIZE - (totalsize + VAR_PROTECTED_BUFFER_SIZE - (FLen - DATA_RECOVE
RY_INDEX)) : VAR_PROTECTED_BUFFER_SIZE, FF);
      size      +=  n;
      totalsize +=  n;  // don't read from recovery record, too !!!

      local_short_crc((unsigned char*)Table, VAR_PROTECTED_BUFFER_SIZ
E, &SubSectorCRC);
      memcpy((unsigned char *)&(ReadIn[buffer_index].Date[Index[buffe
r_index]].buffer[0]), (unsigned char *)&Table, VAR_PROTECTED_BUFFER
_SIZE);
      ReadIn[buffer_index].Date[ Index[buffer_index]].CRC = SubSector
CRC;
```

Recovery

```
    Index[buffer_index]++;
    buffer_index = (buffer_index + 1) % VAR_NON_CONSECUTIVE_NO;

    if((  (n != VAR_PROTECTED_BUFFER_SIZE) && !((i % (VAR_PROTECTED
_BUFFER_NO * VAR_NON_CONSECUTIVE_NO) == 0) && i))
        || (totalsize >= FLen - DATA_RECOVERY_INDEX))
    {
       for(p=0; p < VAR_NON_CONSECUTIVE_NO; p++)
        {
          memset((unsigned char *)&XOR_Rez[p], 0, VAR_PROTECTED_BUF
FER_SIZE);
          memcpy((unsigned char *)&XOR_Rez[p], (unsigned char *)&(R
eadIn[p].Date[0].buffer[0]), VAR_PROTECTED_BUFFER_SIZE);

          for(k=1; k < Index[p]/*VAR_PROTECTED_BUFFER_NO*/; k++)
           for(j=0; j < VAR_PROTECTED_BUFFER_SIZE; j++)
               XOR_Rez[p][j] ^= ReadIn[p].Date[k].buffer[j];

          memset((unsigned char *)&(Protector/*[SectorsNo]*/.recove
ry_[p].buffer[0]), 0, VAR_PROTECTED_BUFFER_SIZE);
          memcpy((unsigned char *)&(Protector/*[SectorsNo]*/.recove
ry_[p].buffer[0]), (unsigned char *)&XOR_Rez[p], VAR_PROTECTED_BUFF
ER_SIZE);
          local_short_crc((unsigned char*)&(Protector/*[SectorsNo]*
/.recovery_[p].buffer[0]), VAR_PROTECTED_BUFFER_SIZE, &SubSectorCRC
);
          Protector/*[SectorsNo]*/.recovery_[p].CRC = SubSectorCRC;
        }
       for(n=0, k=0; k < VAR_PROTECTED_BUFFER_NO; k++)
         for(p=0; p < VAR_NON_CONSECUTIVE_NO; p++)
           Protector/*s[SectorsNo]*/.SubSectorCrc[n++] = ReadIn[p].
Date[k].CRC;

       n=0; for(p=0; p < VAR_NON_CONSECUTIVE_NO; p++) n+=Index[p];
       Protector/*s[SectorsNo]*/.n_subsectors  = n;
       Protector/*s[SectorsNo]*/.n_buffer_size = size;
       WriteCorrectionTable(FF, 1);
       SectorsNo++;
       break;
    }
  }
}

int ReadCorrectionTable(FILE *FF, unsigned int FSize)
{
```

Recovery

```
  fseek(FF, WhereData, SEEK_SET);
  if(ReadProtectStructure(FF, WhereData == FLen + DATA_RECOVERY_IND
EX)) return 1;
  WhereData = ftell(FF);

  fseek(FF, WhereFile, SEEK_SET);
  return 0;
}

int WriteCorrectionTable(FILE *FF, char flush)
{
  WhereFile = ftell(F1);
  fseek(F1, WhereData, SEEK_SET);
  WriteProtectStructure(FF, 1);

  if(WhereFile >= FLen) return 1;

  WhereData = ftell(F1);
  fseek(F1, WhereFile, SEEK_SET);
  return 0;
}

void DeleteHeader(FILE *FF)
{
  rewind(FF);
  ARH_H._recovery_Filesize    = 0;
  ARH_H._recovery_Method      = 0;
  ARH_H._recovery_Compressed = 0;
  if(ZipCloaking) fseek(FF, sizeof(ZIP_H) + ZIP_H.filenamel, SEEK_S
ET);

  //if(fwrite(&ARH_H, 1, sizeof(ARH_H), FF) != sizeof(ARH_H))  Erro
r(7, "");
  if(FWrite_ARH_Header(&ARH_H, FF)) Error(7, ""); // rewritten, has
 a 4 byte CRC built in!
}

int WipeCorrectionTables(FILE *FF)
{ unsigned int trunc;

  Printf(" Scanning for data recovery records, please wait", " ...
", 3);
  trunc = ARH_H._recovery_Filesize;

  DeleteHeader(FF);
```

Recovery

```
  Printf(" Removing recovery records ... ", " ", 2);
  _chsize(fileno(FF), trunc);  // now truncate the file FF (F1) to
the requested file size !
  return 0;
}

int CheckDataRecovery(FILE *FF)
{
  unsigned short CRC1, i, f, k2 = 0, j1 = 0, pp=0;
  unsigned char  DetectionTable[ PROTECTED_BUFFER_SIZE ], tmp[30];

  Sectoros = SectorsK = SectorsBadNo = SectorsRepairedNo = 0;
  for(;;pp++)
   {
     f = ReadCorrectionTable(F1, FLen);
     for(i=0; i < VAR_NON_CONSECUTIVE_NO; i++)
     {
       memcpy((unsigned char *)DetectionTable, (unsigned char*)&(Pr
otector.recovery_[i].buffer[0]), VAR_PROTECTED_BUFFER_SIZE);
       local_short_crc((unsigned char *)DetectionTable, VAR_PROTECT
ED_BUFFER_SIZE, &CRC1);

       sprintf((char*)tmp, " %02d:%02d ", pp, i);
       if((unsigned short)CRC1 == (unsigned short)Protector.recover
y_[i].CRC)
       {
           if(((j1++) % 2 == 0) && (k2 == 0)) Printf(".", "", 3);
       }
       else
       {
         Printf("Recovery record ", " ", 3); Printf((char*)tmp, " "
, 11); Printf(" has failed CRC.", "\n", 3);
         k2++;
       }
     }
     if(f) break;
  }
  sprintf((char*)tmp, " %d ", k2);

  Printf(k2 == 0 ? " All Ok! " : (char*)tmp, " ", 2);
  Printf(k2 == 0 ? " " : " bad recovery records detected", "!", 2);

  return 0;
}
```

Recovery

```
gned int Size, unsigned short *errorIndex) // returns the subsector
 dameged !
{
  unsigned int   i, k=0, errors = 0;
  unsigned short CRC1;
  unsigned char  DetectionTable[ PROTECTED_BUFFER_SIZE ];

  for(i=0; i < Protector.n_subsectors; i++)
  {
    memcpy((unsigned char *)DetectionTable, (unsigned char *)&Table
[(i*VAR_PROTECTED_BUFFER_SIZE)], VAR_PROTECTED_BUFFER_SIZE);
    local_short_crc((unsigned char *)DetectionTable, VAR_PROTECTED_
BUFFER_SIZE, &CRC1);
    if(Protector.SubSectorCrc[i] != CRC1)
    {
        errors = 1;
        errorIndex[i] = 1;
    }
    if(errors == 0)
    {
        sprintf((char *)Tmp__, "%2d:%02d [%X...%X]", SectorI,i, Sec
torI*Protector/*s[SectorI]*/.n_buffer_size +  i*PROTECTED_BUFFER_SI
ZE,  SectorI*Protector/*s[SectorI]*/.n_buffer_size + i*VAR_PROTECTE
D_BUFFER_SIZE);
        Printf(" Sector", " ", 3);
                Printf(Tmp__, " ", 11);
                Printf(" has CRC Ok", "!", 3);
    }
    if(i*VAR_PROTECTED_BUFFER_SIZE >= Size) break;
  }
  return (errors);
}

void Repair(unsigned char *Table, unsigned int bad_subsector, unsig
ned int TableSize)
{
  char OkData[PROTECTED_BUFFER_SIZE];
  char Data  [PROTECTED_BUFFER_NO][PROTECTED_BUFFER_SIZE];

  unsigned int   i, j, k, non_consecutive;
  unsigned short KCRC;

  non_consecutive = bad_subsector % VAR_NON_CONSECUTIVE_NO;

/* primul X = bad_subsector % VAR_NON_CONSECUTIVE_NO;
        X
```

Recovery

```
          X
          X

    Mega Sector

     X          ++++++++    <- Subsector 0
     X          ++++++++    <- Subsector 1
     X          ++++++++    <- .......
     X          ++++++++
     X          ++++++++    <---- BadSector [this is the damaged se
ctor]
     X          ++++++++
     X          ++++++++     ..............
     X          ++++++++
   -----     ------------ ^ ..... . . ... . ...

     XOR_Rez equ : XOR_Rez[i] ^= Table[i];
  */

  for(i=non_consecutive, j=0; i < Protector/*s[NodeID]*/.n_subsecto
rs; i+=VAR_NON_CONSECUTIVE_NO, j++)
  {
     memset(Data[j], 0,  VAR_PROTECTED_BUFFER_SIZE);
     if(i*VAR_PROTECTED_BUFFER_SIZE < TableSize)
         memcpy((unsigned char *)&Data[j], (unsigned char *)&Table[
(i*VAR_PROTECTED_BUFFER_SIZE)], VAR_PROTECTED_BUFFER_SIZE);
  }
  unsigned char XORez[PROTECTED_BUFFER_SIZE];
  memcpy((unsigned char*)XORez, (unsigned char*)&(Protector.recover
y_[non_consecutive].buffer[0]), VAR_PROTECTED_BUFFER_SIZE);
  memset(OkData, 0,  sizeof(OkData));
  for(k=0; k < VAR_PROTECTED_BUFFER_SIZE; k++)
   {
     for(i=0; i < j; i++)
     {
       if(i != (bad_subsector / VAR_NON_CONSECUTIVE_NO))
           OkData[k] ^= Data[i][k];

       // bad_subsector = 11 ==>
       //                          X =  5
       //                          X = 11   <- dar din Data = 1 = (1
1 / 6)
       //                          X = 17 ...
       //
     }
     OkData[k] ^= XORez[k];
  }
```

Recovery

```
  // compute a short crc
  local_short_crc((unsigned char *)OkData, VAR_PROTECTED_BUFFER_SIZ
E, &KCRC);  //pointer to bytes to pump through, number of bytes in
s[] */
  if(Protector/*s[NodeID]*/.SubSectorCrc[bad_subsector] == KCRC)
  {
    // recopy the affected sub-sector, but only if okay, cause othe
rwise, all buffer long
    // might be crased because of a single char wrong

    memcpy((unsigned char *)&Table[(bad_subsector*VAR_PROTECTED_BUF
FER_SIZE)],
           (unsigned char *)&OkData, VAR_PROTECTED_BUFFER_SIZE);
    Print (" ...", " ", 2, "recovered !", " ", 14, " ", " ", 14, "
", " ", 14, -1);
    SectorsRepairedNo ++;
  }
  else  Print (" ...", " ", 2, "unable to recover " , "!", 4, " ",
" ", 4, " ", " ", 0, -1);
}

void Repair()
{ int A = 0;
  unsigned short ErrorIndex[ NON_CONSECUTIVE_NO * PROTECTED_BUFFER_
NO ];

  Printf (" Searching for bad sectors ... ", " ", 2);
  Sectoros = SectorsK = SectorsBadNo = SectorsRepairedNo = 0;

  for(;;)
   {
    if(ReadCorrectionTable(F1, FLen)) break;
    memset(ErrorIndex, 0, sizeof(ErrorIndex));
    memset((unsigned char *)&Final, 0, sizeof(Final));
    fread ((unsigned char *)&Final, 1, Protector.n_buffer_size, F1)
;
    WhereFile = ftell(F1);

    if(DamageDetected(Final, SectorsK, Protector.n_buffer_size, Err
orIndex))
     {
       for(int ko=0; ko < Protector.n_subsectors; ko++)
        {
         if(ErrorIndex[ko] == 1)
          {
```

Recovery

```
            SectorsBadNo++;
            sprintf((char *)Tmp__, "%d:%d [%X...%X]", SectorsK, ko
, Protector.n_buffer_size*SectorsK +  ko*PROTECTED_BUFFER_SIZE,  Pr
otector.n_buffer_size*SectorsK + (ko+1)*VAR_PROTECTED_BUFFER_SIZE);
            Printf(" Sector ", " ",  3);
                        Printf(Tmp__, " ", 11);
                        Printf(" has failed CRC", " ", 3);

            Repair(Final, ko, Protector.n_buffer_size);
          }
          else
          {
            sprintf((char *)Tmp__, "%d:%d [%X...%X]", SectorsK, ko
, Protector.n_buffer_size*SectorsK +  ko*PROTECTED_BUFFER_SIZE,  Pr
otector.n_buffer_size*SectorsK + (ko+1)*VAR_PROTECTED_BUFFER_SIZE);
            Printf(" Sector ", " ", 3);
                        Printf(Tmp__, " ", 11);
                        Printf(" has CRC Ok! ", " ", 3);
           }
         }
       fwrite(Final, 1, Protector.n_buffer_size, F2);
      }
      else fwrite(Final, 1, Protector.n_buffer_size, F2);
      SectorsK++;
     }

   Printf(" Done detecting bad sectors ", "!", 3);


   if(SectorsK == 0) {  Printf(" There was an error : could not rea
d the recovery record, aborting repairing", "!", 12); return; }
   if(SectorsBadNo == 0)  Printf(" No Sectors is damaged, aborting
 repairing", "!", 3);
   else
    {
       char T1[100], T2[100], T3[100];
       sprintf(T1, "%d",     SectorsBadNo); sprintf(T2, "%d",     S
ectorsRepairedNo);.
       sprintf(T3, "%d%c",  (SectorsRepairedNo * 100) / SectorsBadN
o, '%');
       sprintf(Tmp__, "%d", (SectorsRepairedNo + 1) / 2);

           Printf(T1, " ", 11);
           Printf(" damaged subsector(s) reported,", " ", 3);
           Printf(T2, " ", 11);
           Printf(" repaired ", "[", 3);
```

Recovery

```
        Printf(Tmp__, " ",  11);
            Printf(" Kbytes],", " ", 3);
            Printf(T3, "\n",  11);
            Printf(" successfully", "!", 3);
    }

   DeleteHeader(F2);
   Fclose(F2, 0, 0);
  _chsize(fileno(F2), fftrunc);  // ensure the file has the desired
 size !
}

void InitializeProtection(unsigned char met, unsigned int size, cha
r action)
{
  switch(action)
  {
    case 2: Printf (" Testing recovery records", ".", 3); return; b
reak;
    case 1: Printf (" Protecting using", " ", 3); break;
    case 0: Printf (" Recovering using", " ", 3); break;
  }

  //DATA_RECOVERY_INDEX = ZipCloaking ?
    //                    sizeof(ARH_H) + sizeof(ZIP_H) + ZIP_H.fil
enamel
      //                  : 0; //sizeof(ARH_H); //6 -> that extra h
eader
  WhereFile = 0;//DATA_RECOVERY_INDEX;

  switch((metoda = met))
    {
      case 0: VAR_NON_CONSECUTIVE_NO     =   3;
              VAR_PROTECTED_BUFFER_NO    =   5;
              VAR_PROTECTED_BUFFER_SIZE = 512;
              Printf(" 0 level protection", ".", 11);//, "[", 2, "3
 out of 5", 2,  "] ! ", 2);
              break;
      case 1: VAR_NON_CONSECUTIVE_NO     =   5;
              VAR_PROTECTED_BUFFER_NO    =   7;
              VAR_PROTECTED_BUFFER_SIZE = 512;
              Printf(" 1st level protection", ".", 11);//, "[", 2,
"5 out of 7", 2,  "] ! ", 2);
              break;
```

Recovery

```
                VAR_PROTECTED_BUFFER_SIZE = 512;
                Printf(" 2nd level protection", ".", 11);//, "[", 2,
"7 out of 11", 2,  "] ! ", 2);
                break;
        case 3: VAR_NON_CONSECUTIVE_NO    =  11;
                VAR_PROTECTED_BUFFER_NO   =  21;
                VAR_PROTECTED_BUFFER_SIZE = 512;
                Printf(" 3rd level protection", ".", 11);//, "[", 2,
"11 out of 21", 2,  "] ! ", 2);
                break;
    }
}
```

Solid

```
/*****************************************************************
***********
 *
          *
 *             The very SOLID stuff xtremly useful procedures
          *
 *
          *
 *           From Xtreme 106 (DLL) package by Sabin, Belu (c) 1999
          *
 *
          *
 *****************************************************************
***********/

#pragma pack (1)

#include <io.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <direct.h>
#include <malloc.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <windows.h>

#include "crc.h"
#include "util.h"
#include "solid.h"
#include "trees.h"
#include "markov.h"
#include "deflate.h"
#include "console.h"

#pragma pack (1)
#include "exports.h"
#include "zipheads.h"
#include "c:\progra~1\micros~2\myproj~1\xtremeall\xconsole\interf.h
"

#pragma pack (1)
// it must have the pragma pack 1 here ....
__declspec( dllexport ) int GetSolidArchiveFileList(char *archive_n
ame, struct XConsoleLista *archive_name_list);
```

Solid

```
SolidFileType *SolidArray   = NULL;
SolidFileType *SolidArray2  = NULL;

int             SolidList = 0, Default = 13;
unsigned int    Solid_ALLOCATED = 3000;
unsigned long   TotalSolidSize;
unsigned int    SolidK = 0, SolidN = 0,  Solid_CRC;

extern struct  TableData configuration_table[6];

int  SolidInit()
{
    if(SolidArray == NULL)  SolidArray = (struct SolidFileType *)ma
lloc(sizeof(struct SolidFileType)*Solid_ALLOCATED);
    Solid_CRC = INIT_CRC_VALUE;
    return 0;
}

int __cdecl SolidCompare(const void  *a, const void  *b)
{
        char Name1[_MAX_PATH];
        char Name2[_MAX_PATH];

        if(!((struct SolidFileType *)a)->ext[0] && !((struct SolidF
ileType *)b)->ext[0])
        {
          strcpy(Name1, ((struct SolidFileType *)a)->RealName);
      if(strchr(Name1, '\\'))
          {
       _splitpath(Name1, NULL, NULL, Name1, NULL);
          }
      strcpy(Name2, ((struct SolidFileType *)b)->RealName);
      if(strchr(Name2, '\\'))
          {
       _splitpath(Name2, NULL, NULL, Name2, NULL);
          }
      return stricmp(Name1, Name2);
        }
        else
        if(stricmp(((struct SolidFileType *)a)->ext, ((struct Solid
FileType *)b)->ext) == 0)
        {
          strcpy(Name1, ((struct SolidFileType *)a)->RealName);
      if(strchr(Name1, '\\'))
```

Solid

```
        }
      strcpy(Name2, ((struct SolidFileType *)b)->RealName);
      if(strchr(Name2, '\\'))
          {
       _splitpath(Name2, NULL, NULL, Name2, NULL);
          }
     return stricmp(Name1, Name2);
        }
       else return stricmp(((struct SolidFileType *)a)->ext, ((str
uct SolidFileType *)b)->ext);

}

void UpdateSolidArray(unsigned int SolidKa)
{
  if(SolidKa > Solid_ALLOCATED - 10)
   {
      Solid_ALLOCATED += 2000;
      SolidArray = (struct SolidFileType *)realloc(SolidArray,
                          _msize(SolidArray) + (2000 * sizeof(stru
ct SolidFileType)));

      if (SolidArray == NULL)
      {
          printf("No memory for registering %d more files", Solid_A
LLOCATED);
          exit(0);
      }
   }
}

void SolidFillTheName(unsigned int J, char *name, unsigned int size
, unsigned long time_date, unsigned char atrib)
{
  char drive[_MAX_DRIVE];
  char dir[_MAX_DIR];
  char fname[_MAX_FNAME];
  char ext[_MAX_EXT];

  _splitpath(name, drive, dir, fname, ext );
   UpdateSolidArray(J);
   memset(SolidArray[J].RealName, 0, _MAX_PATH);
   strcpy(SolidArray[J].RealName, name);

   SolidArray[J].size      = size;
   SolidArray[J].time_date = time_date;
```

Solid

```
    SolidArray[J].atrib      = atrib; // to compress

    TotalSolidSize += size;
    memset(SolidArray[J].ext,  0, 18);
    if(ext[0]) strncpy(SolidArray[J].ext, (char *)&ext[1], 18);
}

int Solid_Register_Files(char *Director)
{
  struct _finddata_t Block;
  char    TmpDir[_MAX_PATH],  Tmp [_MAX_PATH], fname [_MAX_FNAME],
ext [_MAX_EXT], Coco[200];
  char    drive [_MAX_DRIVE], dir [_MAX_DIR];
  char    fname2[_MAX_FNAME], ext2[_MAX_EXT];
  long    Done;

  ComplexSplit(Director, drive, dir, fname, ext); // cred ca acest
ComplexSplit imi face directoarele cu lower !!!
  memset(Director, 0, sizeof(Director));
  strcat(Director, drive);
  strcat(Director, dir);
  strcat(Director, "*.*");  // deci nu avem extensie la directoare
ca de exmplu : XtremeDLL.a06

  strcpy(ext, ".*");

  if((Done = _findfirst(Director, &Block)) == -1L)  {  Error(12, Di
rector); return 0; } // no files
  do{
      if((Block.attrib & 0x10) == 0x10)
       {
          memset(TmpDir, 0, sizeof(TmpDir));
          strncpy(TmpDir, Director, strlen(Director) - 3/*strlen(fn
ame) - strlen(ext)*/);

          strcat(TmpDir, Block.name);
          strcat(TmpDir, "\\");
          strcat(TmpDir, fname);
                  strcat(TmpDir, ext);
          if(strstr(TmpDir, ".\\") == NULL)
          {
           if(Recursiv)
                   {
                           strcpy(Tmp, TmpDir);
                           Tmp[strlen(Tmp)-3] = 0;
```

Solid

```
                sprintf(Coco, "%-36s", short_str(Tmp, 35));
                Printf(" Scanning sub-directory ", "", 3);
                Printf(Coco, "\r", 2);

                    Solid_Register_Files(TmpDir);
             }
            }
       }
       else
       {
        _splitpath(Block.name, NULL, NULL, fname2, ext2);
         if(Verify(fname2, fname, strlen(fname2), strlen(fname)) &&
            Verify(ext2,     ext, strlen(  ext2), strlen(  ext)))
         {
            memset(TmpDir, 0, sizeof(TmpDir));
            strncpy(TmpDir, Director, strlen(Director) - 3/* *.* str
len(fname)  - strlen(ext)*/ );
            strcat(TmpDir, Block.name);

                    if(Block.size)
            {
              SolidFillTheName(SolidN++, TmpDir, Block.size, Block.t
ime_write,  (unsigned char)Block.attrib); // don't register zero len
gth file!
            }
         }
       }
  }while(_findnext(Done, &Block)  == 0);

  _findclose(Done);
  return 1;
}

void SolidExtract(void)
{
        int uu = ftell(F1);

    SolidK = 0;
    memset(InFile, 0, _MAX_PATH);
    strcpy(InFile, SolidArray[0].RealName);
    TotalSum = SolidArray[SolidK++].size;

    if((Sterg == 1 || Actualizez == 1 || Extrag == 1) && (SolidArra
y[0].selected == 1))  write_melt = 0;
    else write_melt = 1;
```

Solid

```
    if(write_melt)  OpenFile(InFile, SolidArray[0].time_date, Solid
Array[0].atrib);
    inflate();
}

int SolidMelt(void)
{
    SolidInit();
    if(Read_Solid_Header()) return 1;
    SolidExtract();
    return 0;
}

void Register_File_List(char *FileMask)
{ char File[_MAX_FNAME];
  char ext [_MAX_EXT];
  char Surs[_MAX_PATH];

  memset(Surs, 0, sizeof(Surs));
  strcpy(Surs, FileMask);
  if(Recursiv)
   {
     Print(" Analyzing directory ", "'", 3, short_str(FileMask, 49
), "'", 2, "", "\n", 0, "", "", 0, -1);

     memset(File, 0, _MAX_FNAME);
         memset(ext,  0, _MAX_EXT);
     _splitpath(FileMask, NULL, NULL, File, ext);

     if(File[0] == 0 || ext[0] == 0)
     {
       if(Surs[strlen(Surs) - 1] != '\\') strcat(Surs, "\\");
       strcat(Surs, "*.*");
     }
   }
  else Printf(" Analyzing ", "...", 3);
  Solid_Register_Files(Surs);
}

void Register_Single_File(char *full_filename)
{
  int fh, result;
  struct _stat buffer;

  if( (fh = _open(full_filename, _O_RDONLY, _S_IREAD))  != -1 )
  {
```

Solid

```
    result = (int)(_filelength(fh));
   _fstat(fh, (struct _stat *)&buffer);

        if(buffer.st_size) // only if size is non zero please !
        {
      SolidFillTheName(SolidN++, full_filename, buffer.st_size, bu
ffer.st_mtime,
                                (unsigned char)(GetFileAttributes(f
ull_filename)));
        }
    _close(fh);
  }
}

void SolidFreeze(char *FileMask)
{
    char Surs[_MAX_PATH];
    char Theatre[_MAX_PATH + _MAX_PATH];
        unsigned int ik;

    SolidInit();
        SolidK = 1;    SolidN = 0;   TotalSolidSize = 0;

    memset(Surs, 0, sizeof(Surs));
    strcpy(Surs, FileMask);

        //
        //  The RealName and Archive Name strategy is 'xtremely' go
od. We can basically read from
        // any directory (since we have two different paths in our
interface), and, that's
        // the most interesting of all, we can even create fake dir
ectories inside the archive
        // like the UC2's 'Archive directory' .... It will be imple
mented in ADD, and then
        // I would like to get rid of that aweful(but usefull, till
 now) SolidArray2. Thank you

    for(ik=0; ik < MyXtrStructure.MaxDirectories; ik++)
        {
            memset(Theatre, 0 ,sizeof(Theatre));
        if(MyXtrStructure.Sursele[ik].Path1Or2 ?
                    MyXtrStructure.Path1[0] :
                    MyXtrStructure.Path2[0])
                    {
                      strcpy(Theatre, (MyXtrStructure.Sursele[ik].Pa
```

Solid

```
th1Or2 ?
                                        MyXtrStructure.Path1 : MyXtrStruct
ure.Path2));

                        CheckSlash(Theatre);  //  if((strlen(Theatre))
 && (Theatre[strlen(Theatre) - 1] != '\\')) strcat (Theatre, "\\");
                      }

                  strcat (Theatre, MyXtrStructure.Sursele[ik].Name);

                  if(MyXtrStructure.Sursele[ik].File) Register_Singl
e_File(Theatre); // if file size is not zero!
         else                                        Register_File_List(The
atre);
         }

    if(Recursiv) Printf(" Directories scanned.
                    ", "\n\n", 3);
    if(SolidN)
        {
      if(Recursiv) Printf(" Analysing ...", "\n", 3);

      sprintf(tmp, "%d", SolidN);
          Print(" Sorting", " ",  3, tmp, " ", 14, SolidN > 1 ? "fi
les ... " : "file.",  "\n",  3, "", "", 4, -1);

      qsort((struct SolidFileType *)SolidArray,  (int)SolidN, sizeo
f(struct SolidFileType), SolidCompare);
          Printf(" Freezing solid header ...", "\n ", 3);

      if(Write_Archive_Header()) return;
          if(Write_Solid_Header())   return;

      sprintf(tmp, "'%s'", short_str( FileMask /*Sursa[0].Name*/, 6
5));
      Print(" Freezing files", " ", 3, tmp, "\n", 2, "", "", 0, "",
 "", 0, -1);

          PackFile(SolidArray[SolidK-1].RealName, SolidArray[SolidK
-1].size);
        }
    Write_Solid_CRC();
}

int TestSolidArchive(void)
{
```

Solid

```
  unsigned long CatAre = _filelength(_fileno(F1));
  unsigned long Undere = 0;

  SolidInit();
  if(Read_Solid_Header()) return 1;

  SolidK = 0;
  memset(InFile, 0, _MAX_PATH);
  strcpy(InFile, SolidArray[0].RealName);
  TotalSum = SolidArray[SolidK++].size; // first file's size

  for(; Read_Block() == 0;)
   {
     Undere = ftell(F1);
     ratio = IntoarceProcent(Undere, CatAre);
     sprintf(tmp, "%35d.%d%c", ratio / 10, ratio % 10, '%');
     Print(" Testing solid archive", " ", 2, Destination, " ", 3, t
mp, "\r", 2, "", "", 0, (ratio / 10));
     if(restbits > 8) break;
   }
 sprintf(tmp, "%-56s", " ");

 if(TestedOk)
 {
       Printf(" Solid archive is Ok !", "\n", 2);
  //Print("", "", 2, "", "", 3, tmp, "\n", 2,  "", "", 0, -1);
       Printf("   ", "\n", 2);
    sprintf(tmp, " %d", SolidN);
    Print(tmp, " ",  10, " files processed.", "\n",  2, "", "", 0,
"", "", 0, -1);
 }
 else
 {
       Printf(" ", "\n", 2);
    Printf(" Solid archive seemes to have BAD CRC .... !", "\n", 2)
;
       Error(2, "");
 }

 return 0;
}

int UpdateSolidArchive(char *files, char _to_solid_or_normal)
{
 char temporaryFilename[_MAX_PATH], temp[_MAX_PATH];
 unsigned int i,  Res = 0, j, ToReplace = 0;
```

Solid

```
 if(Read_Archive_Header(1)) { Error(11, ""); return 1; }
 SolidInit();
 if(Read_Solid_Header()) return 1;

 Print(" Scanning for freezed", " '", 3, files, ". ", 2, "',", " ",
3, "", "", 0, -1);
 FileSelection();
 SolidArray2 = (struct SolidFileType *)malloc((SolidN + 3) * sizeof
(struct SolidFileType));
 if(SolidArray2 == NULL) return 1;

 for(i=0, j=0; i < SolidN; i++)
 {
         if(SolidArray[i].selected == 1)
           ToReplace++;  // easier, isn't it ?
         else
         {
       Res++;
       memcpy((unsigned char *)&SolidArray2[j++], (unsigned char *)
&SolidArray[i], sizeof(struct SolidFileType));
         }
 }

 sprintf(temp, " %d files will be replaced.", ToReplace);
 Printf(temp, "\n", 2);

 if(ToReplace < SolidN)
 {
   GetTemporaryDirectory(extractionDestination);
   Print(" Extracting to temporary directory ", "'", 3, extractionD
estination, "", 2, "' ", " ", 0, "", "", 0, -1);
   SolidExtract();
 }
 Fclose(F1, 0, 0);  Fclose(F2, 0, 0);

 Print(" Deleting previously packed files,", " '", 3, files, "", 2,
 "' ", "", 0, "", "", 0, -1);

 for(i=0; i < SolidN; i++)
  {
    if(SolidArray[i].selected)
        {
           char tmp2[_MAX_PATH];
           sprintf(tmp2, " %-63s" , short_str(SolidArray[i].RealNam
e, 63));
```

Solid

```
        Print(tmp2, "", 2, " deleted", " ", 3,"", "", 0, "", "", 0,
-1);
         }
 }

 SolidN = 0;
 for(unsigned int ik=0; ik < MyXtrStructure.MaxDirectories; ik++)
 {
   memset(temp, 0 ,sizeof(temp));
   strcpy(temp, (MyXtrStructure.Sursele[ik].Path1Or2 ?
                    MyXtrStructure.Path1 : MyXtrStructure.Path2));
   CheckSlash(temp);
   strcat (temp, MyXtrStructure.Sursele[ik].Name);
   if(MyXtrStructure.Sursele[ik].File)  Register_Single_File(temp);
 // if file size is not zero!
   else                                 Register_File_List(temp);
 }

 // maybe if SolidN == 0 to return 1;
 for(i=0; i < SolidN; i++)  SolidArray[i].selected = 0;
 for(i=0; i < Res; i++)
 {
    UpdateSolidArray(SolidN);
    memcpy((unsigned char*)&SolidArray[SolidN], (unsigned char*)&So
lidArray2[i], sizeof(struct SolidFileType));
    SolidArray[SolidN].selected = 2;  // files that need extra addi
tion!
    SolidN++;
 }
 if(SolidArray2) free(SolidArray2);

 if(CreateTemporaryArchive(extractionDestination, temporaryFilename
)) return 1;
 SolidRePack(SolidN, extractionDestination, 1);

 Printf(" Deleting temporary files",  "", 3);

 for(i=0; i < SolidN; i++)
 { // please do not delete the line with == 2, cause this will move
 files to archive !
   if(SolidArray[i].selected == 2) _unlink_path(SolidArray[i].RealN
ame, 1);
 }
 Fclose(F1, 0, 0); Fclose(F2, 0, 0);

 GetCurDir(temp);
```

Solid

```
 if(strstr(Destination, ":\\") == NULL) strcat(temp, Destination);
 else strcpy(temp, Destination);

 _unlink(temp);
 rename(temporaryFilename, temp);
 return 0; // everything went okay !
}

int SolidDeleteFiles(char *f)
{
 char temp[_MAX_PATH], temporaryFilename[_MAX_PATH];
 unsigned int i, Res = 0, j;

 SolidInit();
 if(Read_Solid_Header()) return 1;

 for(i=0; i < SolidN; i++)
 {
        if(SolidArray[i].selected == 1) Res++;  // easier, isn't i
t ?
 }

 sprintf(temp, " %d file(s) will be deleted.", Res);

 Printf(temp, "\n", 2);
 if(Res == 0) return 1;
 if(Res != SolidN)
 {
   GetTemporaryDirectory(extractionDestination);
   Print(" Extracting to temporary directory ", "'", 3, extractionD
estination, "'", 2, "", "", 0, "", "", 0, -1);
   SolidExtract();
 }
 Fclose(F1, 0, 0);  Fclose(F2, 0, 0);

 Print(" Deleting the requested files, ", "'", 3, f, "", 2, "' ", "
", 0, "", "", 0, -1);

 for(i=0; i < SolidN; i++)
  {
    if(SolidArray[i].selected)
        {
       char tmp2[_MAX_PATH];
           sprintf(tmp2, " %-63s" , short_str(SolidArray[i].RealNam
e, 63));
       Print(tmp2, "", 2, " deleted", " ", 3,"", "", 0, "", "", 0,
```

Solid

```
-1);
        }
 }

 if(Res != SolidN)
 {
   SolidArray2= (struct SolidFileType *)malloc((SolidN + 1) * sizeo
f(struct SolidFileType));
   if(SolidArray2 == NULL) return 1;
   for(i=0, j=0; i < SolidN; i++)
   {
     if(SolidArray[i].selected == 0)
         {
       memcpy((struct SolidFileType *)&SolidArray2[j++], (struct So
lidFileType*)&SolidArray[i], sizeof(struct SolidFileType));
         }
   }
   for(i=0; i < j; i++)
   {
       memcpy((struct SolidFileType*)&SolidArray[i], (struct SolidF
ileType*)&SolidArray2[i], sizeof(struct SolidFileType));
   }

   if(CreateTemporaryArchive(extractionDestination, temporaryFilena
me)) return 1;
   SolidRePack(j, extractionDestination, 0);

   Printf(" Deleting temporary files", "\n", 3);
   for(i=0; i < SolidN; i++)  _unlink_path(SolidArray[i].RealName,
1);
 }
 Fclose(F1, 0, 0);

 if(ZipCloaking)  // ????
 {
   for(i=0; i < SolidN; i++)
   {
      memcpy((struct SolidFileType*)&SolidArray[i], (struct SolidFi
leType*)&SolidArray2[i], sizeof(struct SolidFileType));
   }
   Write_Solid_CRC();
 }

 Fclose(F2, 0, 0);

 //GetCurDir(temp);
```

Solid

```
 //strcat(temp, Destination); // not good anymore ....

 GetCurDir(temp);
 if(strstr(Destination, ":\\") == NULL) strcat(temp, Destination);
 else strcpy(temp, Destination);

 _unlink(temp);
 if(SolidArray2) free(SolidArray2);
 if(Res == SolidN)  return 0;
 rename(temporaryFilename, temp);
 return 0; // all went okay !
}

void SolidRePack(unsigned int maxim, char *fromWhere, char _update)
{
 char temp[_MAX_PATH];
 unsigned int i;

 SolidN = maxim; SolidK = 1; TotalSolidSize = 0;
 qsort((struct SolidFileType *)SolidArray,  (int)SolidN, sizeof(str
uct SolidFileType),  SolidCompare);

 if(Write_Archive_Header()) return;
 if(Write_Solid_Header())   return;

 Printf(" Repacking files", " \n", 3);

 for(i=0; i < SolidN; i++)
  {
     if((_update == 0 )  || (_update == 1 && SolidArray[i].selected
 == 2))
     {
       memset(temp, 0, _MAX_PATH);
       strcpy(temp, fromWhere);
       strcat(temp, SolidArray[i].RealName);
       strcpy(SolidArray[i].RealName, temp);
     }
     TotalSolidSize += SolidArray[i].size;
  }
 PackFile(SolidArray[SolidK-1].RealName, SolidArray[SolidK-1].size)
;
}

int Read_Solid_Header(void)
{ struct Solid_XTREME_Header SolidXTREME_H;
  unsigned int    SizeS = 0, x= sizeof(SolidXTREME_H), Sin, Sout, X
```

Solid

```
;
  unsigned char  *Undefined, *Undefined2;
  unsigned short  crc1, crc2, rezult = 0;

  if(ZipCloaking)  return (Read_ZIP_ARH_Files());

  SolidN = getw(F1);
  TotalSolidSize = 0;

  sprintf(tmp, " Melting solid header [%d files] ", SolidN);
  Printf (tmp, " ", 3);
  InitializeMarkovStates();

  X = (sizeof(struct Solid_XTREME_Header) + _MAX_PATH) * SolidN;
  memory_request_alloc(unsigned char, Undefined,  X);
  memory_request_alloc(unsigned char, Undefined2, X);

  if(Undefined == NULL || Undefined2 == NULL) return 1;
  memset(Undefined, 0, X);

  if(fread(&crc1, sizeof(crc1), 1, F1) < 1)  Error(6, "");
  if(fread(&Sout, sizeof(Sout), 1, F1) < 1)  Error(6, "");
  if(fread(Undefined, 1, Sout, F1) != Sout)  Error(6, "");
  MarkovDecode(Sout, Undefined, Undefined2, &Sin);
  local_short_crc(Undefined, Sout, &crc2);
  if((unsigned short)crc1 == (unsigned short)crc2)
  {
          Printf(" Result : [CRC Ok] ", "\n", 3);
          for(unsigned int i=0; i < SolidN; i++)
          {
        UpdateSolidArray(i);
         //      if(i % (SolidN / 23 == 0 ? 1 : SolidN / 23) == 0)
 Printf(".", 15);

        memcpy(&SolidXTREME_H, (unsigned char*)&Undefined2[SizeS],
x);
        memset(InFile, 0, _MAX_PATH);
        memcpy(InFile, (unsigned char*)&Undefined2[SizeS + x], Soli
dXTREME_H._NamLen);

        SizeS += x + SolidXTREME_H._NamLen;
        TotalSolidSize    += SolidXTREME_H._SizeUnCom;
        SolidFillTheName(i, InFile, SolidXTREME_H._SizeUnCom, Solid
XTREME_H._Time_Date, SolidXTREME_H._Atrib);
          }
          rezult = 0;
```

Solid

```
    }
  else
  {
          Printf(" Result : [CRC Failed]", "\n", 3);
          rezult = 1;
  }

  if(Undefined2 != NULL) { free(Undefined2); Undefined2 = NULL; }
  if(Undefined  != NULL) { free(Undefined);  Undefined  = NULL; }
  De_Init_Buf(); // Deinitialize Markov States

  if(rezult == 1)
  {
          Error(6, " Solid header seems a little bit damaged .... "
);
          return 1;
  }

  return 0;
}

int Write_Solid_Header(void)
{ short ratio;
  unsigned char *Undefined, *Undefined2;
  struct Solid_XTREME_Header SolidXTREME_H;
  unsigned int SizeS = 0, x = sizeof(SolidXTREME_H), Sout, X;
  unsigned short crc;

  char TmpFile[_MAX_PATH], tmpp[300];

  if(ZipCloaking) return 0; // no need for another zip header, here
 !

  putw(SolidN, F2);
  InitializeMarkovStates();  // initialize the Markov model
  X = (sizeof(struct Solid_XTREME_Header) + _MAX_PATH) * SolidN;
  memory_request_alloc(unsigned char, Undefined,  X);
  memory_request_alloc(unsigned char, Undefined2, X);
  if(Undefined == NULL || Undefined2 == NULL) return 1;
  memset(Undefined, 0, X);

  for(unsigned int i=0;i<SolidN;i++)
      {
     SolidXTREME_H._Time_Date  = SolidArray[i].time_date;
     SolidXTREME_H._SizeUnCom  = SolidArray[i].size;
     SolidXTREME_H._Atrib      = SolidArray[i].atrib;
```

Solid

```
        SolidXTREME_H._NOT_USED1   = 32;
        SolidXTREME_H._NOT_USED2   = 32;

            memset(InFile,   0, _MAX_PATH);
            memset(TmpFile, 0, _MAX_PATH);
        strcpy(InFile, SolidArray[i].RealName);
            strcpy(TmpFile, (char *)&InFile[strstr(InFile, ":\\") ? 3
 : 0]);
        SolidXTREME_H._NamLen  = (strlen(TmpFile));

        memcpy((unsigned char*)&Undefined[SizeS], &SolidXTREME_H, x);
        memcpy((unsigned char*)&Undefined[SizeS + x], TmpFile, SolidX
TREME_H._NamLen);

        SizeS += x + SolidXTREME_H._NamLen;
      }
      MarkovEncode(SizeS, Undefined, Undefined2, &Sout);
      local_short_crc(Undefined2, Sout, &crc);
      if(fwrite(&crc,  sizeof(crc),  1, F2) < 1)  Error(5, "");
      if(fwrite(&Sout, sizeof(Sout), 1, F2) < 1)  Error(5, "");
      if(fwrite(Undefined2, 1, Sout, F2) < Sout)  Error(5, "");

      ratio = IntoarceProcent(Sout, SizeS);
      sprintf(tmpp," Solid header freezed [%d.%d%c]", ratio / 10, ratio
 % 10, '%');
      if(Repack == 0)           Printf(tmpp, "\n", 2);
      if(Undefined2 != NULL) { free(Undefined2); Undefined2 = NULL; }
      if(Undefined  != NULL) { free(Undefined);  Undefined  = NULL; }
      De_Init_Buf(); // Deinitialize Markov States
      return 0;
    }

    int Write_Solid_CRC(void)
    { char  T[30];
      short ratio;

      ratio = IntoarceProcent(TotalWrit, TotalSolidSize);
      sprintf(T, "%3d.%d%c", ratio / 10, ratio % 10, '%');
      Printf(" Updating solid archive header.", "\n", 3); // for CRC !
      if(ZipCloaking)
      {
         WriteZipHeaders(&TotalWrit);
      }
      Print(" Solid archive ratio", " ", 3, T, "", 2, "", "", 0, "", ""
     , 0, -1);
      Printf("\n", "", 0);
```

Solid

```
  return 0;
}

/*****************************************************************
*******************
 * BEGIN :  This function is only for DLL export use only. Do not t
ry it inside the DLL.
 *****************************************************************
*******************/

int GetSolidArchiveFileList(char *archive_name, struct XConsoleList
a *archive_name_list)
{
  int Rez = 0;
  unsigned int k;
  int ii = sizeof(struct XConsoleLista);

  if((F1 = fopen(archive_name, "rb+")) == NULL) return 1;
  if(Read_Archive_Header(0))  return 1; // the zero parameter does
not allow memory allocation ...
  SolidInit();
  if(Read_Solid_Header()) Rez = 1;
  Fclose(F1, 0, 0);

  // now for the 0 .. SolidN-1 will just copy the files and that's
all.
  for(k=0; k<SolidN; k++)
  {
    memset((unsigned char *)&(archive_name_list[k]), 0, sizeof(str
uct XConsoleLista));
        strcpy(archive_name_list[k].Name, SolidArray[k].RealName);
    archive_name_list[k].FSize    = SolidArray[k].size;
    archive_name_list[k].DateTime = SolidArray[k].time_date;
  }

  return Rez;
}
/*****************************************************************
*******************
 * END   : This function is only for DLL export use only. Do not tr
y it inside the DLL.
 *****************************************************************
*******************/

int SolidListFiles(char *files)
{
```

Solid

```
 char files1[_MAX_PATH], dir1[_MAX_PATH], ext1[_MAX_PATH];
 char files2[_MAX_PATH], dir2[_MAX_PATH], ext2[_MAX_PATH];
 unsigned int  i, Res = 0;
 unsigned long compressed_all = 0, uncompressed_all = filelength(fi
leno(F1));

 SolidInit();
 if(Read_Solid_Header()) return 1; // this function is perfect. It
filles up a structured array, ready to be printed out ...

 sprintf(tmp, " Name %54s  %10s %s", "Original", "Compressed", "Rat
io");
 sprintf(tmp2, " ----------------------------------------------------
----------------------------");

 Printf(tmp,  "\n",  3);
 Printf(tmp2, "\n", 15);

 Split(files[0] == 0 ? "*.*" : files, NULL, dir1, files1, ext1);
 for(i=0; i < SolidN; i++)
  {
     Split(SolidArray[i].RealName, NULL, dir2, files2, ext2);
     if( Verify(files2, files1, strlen(files2), strlen(files1))    &
&
         Verify(ext2,     ext1, strlen(  ext2), strlen(  ext1))    &
&
              ((strstr(dir2, dir1) == dir2 && (Recursiv))     ||
              ((stricmp(dir2, dir1) == 0)    && (Recursiv == 0))   ||
 files[0] == 0 ))
        {
            Res++;
            sprintf(tmp,   " %-49s", short_str(SolidArray[i].RealNam
e, 49) );
            sprintf(tmp2, "%10d", SolidArray[i].size);
            compressed_all += SolidArray[i].size;
            Printf(tmp, " ", 2);
                        Printf(tmp2, "\n", 3);
        }
 }
 sprintf(tmp2," ----------------------------------------------------
----------------------------");
 sprintf(tmp, " %d ", Res);

 memset(files1, 0, sizeof(files1));memset(files2, 0, sizeof(files2)
);memset(dir1, 0, sizeof(dir1));memset(dir2, 0, sizeof(dir2));
 SprintfBigNumber(dir1,   compressed_all);sprintf(files1, " %60s ",
```

Solid

```
 dir1);
 SprintfBigNumber(dir2, uncompressed_all);sprintf(files2, " %10s ",
 dir2);

 ratio = IntoarceProcent(uncompressed_all, compressed_all);
 sprintf(ext1, "%3d.%d%c", ratio/10, ratio%10, '%');

 Printf(tmp2,    "", 15);
 Printf(files1, "", 3);        // files sizxe uncompressed
 Printf(files2, "", 3);        // file size compressed
 Printf(ext1,    "\n", 3);     // file ratio then \n
 Printf(tmp,     "", 10);      // 5
 Printf("file(s).", "\n", 3); // files processed!

 return 0;
}
```

Trees

```
/***************************************************************************
***********
 *
          *
 *              Trees compression/decompression procedures
          *
 *
          *
 *            Dynamic Markov / Dynamic Huffman / XXXAri
          *
 *
          *
 *          From Xtreme 106 (DLL) package by Sabin, Belu (c) 1999
          *
 *
          *
 ***************************************************************************
***********/
#pragma pack (1)

#include   <stdio.h>
#include   <stdio.h>
#include   <string.h>

#include   "crc.h"
#include   "util.h"
#include   "huffman.h"
#include   "solid.h"
#include   "trees.h"
#include   "markov.h"
#include   "console.h"
#include   "deflate.h"
#include   "Contextual.h"

struct Huff_Block Huffman;

unsigned int      CRC, HuffmanBlockCRC;
unsigned char     maska, flag, restbits;
unsigned short    D_REST_BITS;
unsigned short    bits_buf;   // Output buffer. bits are inserted s
tarting at the bottom (least significant
short             bits_len;   //  Number of valid bits in bi_buf.  A
ll bits above the last valid bit are 0.


#define  PutW(c) PutC((c >> 8) & 0x00ff); \
```

Trees

```
                PutC((c >> 0) & 0x00ff); \

#define  GetC()  getc(F1); // if used somewhere else remember this
: TotalWrit -- !!!!

#define  GetW(c) c=0;          \
                 c=GetC();     \
                 c<<=8;        \
                 c|=GetC();    \
                 TotalWrit-=2;

// this array is used to get only the number of the required bits.

static unsigned short B_MASK[] =
  {
    0x0000, 0x0001, 0x0003, 0x0007,
    0x000f, 0x001f, 0x003f, 0x007f,
    0x00ff, 0x01ff, 0x03ff, 0x07ff, 0x0fff, 0x1fff, 0x3fff, 0x7fff,
 0xffff
 };

void Tree_Start()
{
  flag = 0;
  maska = 1;
  restbits = 0;
  BitBufC = 0;
  LitBufC  = LenBufC = DistBufC = RestBufC1 = RestBufC2 = DistBufC
= 0U;

  bits_buf = 0;
  bits_len = 0;
}

#define MARKOV_TYPE                              0
#define HUFFMAN_TYPE                             1
#define CONTEXTUAL_TYPE                          2
#define BLOCK_TYPE_COMPRESSION  CONTEXTUAL_TYPE

int FrozenBuffer(unsigned int len, unsigned char *Bufferu, char typ
e_of_block)
{
  if (fwrite(&len, sizeof(len), 1, F2) < 1) Error(3, "");
  updcrc((unsigned char *)Bufferu, (unsigned int)len, &CRC);

  //
```

Trees

```
  // We need a call on the function for a better update of the mess
age
  // in the window, because it just freezes and it's not quite ok
  //
  //

  Print(" Frosting blocks, please wait ", " ", 3, " ", " " , 2, "",
 "", 0, "", "", 0,  -1);

  switch(type_of_block)
  {
    case  HUFFMAN_TYPE     : EncodeBuffer(len, Bufferu, OutBuf, &out
count);              break;
    case  MARKOV_TYPE      : MarkovEncode(len, Bufferu, OutBuf, &out
count);              break;
    case  CONTEXTUAL_TYPE : ContextualEncodeBuffer (len, Bufferu, O
utBuf, &outcount); break;
  }
  if(outcount < len)
  {
     if (fwrite(&outcount, sizeof(outcount), 1, F2) < 1) Error(3, "
");
     if (fwrite(OutBuf, 1, outcount, F2) < outcount)     Error(3, "
");
     TotalWrit += outcount + 2*sizeof(len);
  }
  else
  {
     if (fwrite(&len, sizeof(len), 1, F2) < 1)           Error(3, "
");
     if (fwrite(Bufferu, 1, len, F2) < len)              Error(3, "
");
     TotalWrit += len + 2*sizeof(len);
  }

  return (outcount < len);
}


void MeltBuffer(unsigned int *len, unsigned char *Bufferu, char Typ
e, char type_of_block)
{
  if(fread(len, sizeof(*len), 1, F1) < 1)           Error(4, "");
  if(fread(&outcount, sizeof(outcount), 1, F1) < 1) Error(4, "");
  if(fread(Bufferu, 1, outcount, F1) != outcount)   Error(4, "");
  int u= ftell(F1);
```

Trees

```
  if(outcount > LITBUF + 1000)
  {
          Error(4, "");// access violation here, requesting help
          // it will exit anyway, so you do not have to do an emerg
ency exit....
  }

  TotalWrit-= outcount + 2*sizeof(len);
  unsigned int last_out = *len;


  if(Type)
  {
    switch(type_of_block)
     {
        case HUFFMAN_TYPE    : DecodeBuffer(*len, Bufferu, OutBuf,
&outcount);           break;
        case MARKOV_TYPE     : MarkovDecode(*len, Bufferu, OutBuf,
&outcount);           break;
        case CONTEXTUAL_TYPE : ContextualDecodeBuffer(*len, Bufferu
, OutBuf, &outcount); break;
     }
     updcrc((unsigned char *)OutBuf, type_of_block ? outcount : (un
signed int)last_out, &CRC);
     memcpy(Bufferu, OutBuf, type_of_block ? outcount : last_out);
  }
  else
  {
     updcrc((unsigned char *)Bufferu, (unsigned int)outcount, &CRC)
;
  }
}

void Send_Block(char end_of_block)
{
  unsigned int  i;
  unsigned long Where1, Where2;
  int u= ftell(F2);

  if(maska != 1)   BitBuf[BitBufC++] = flag;
  CRC = INIT_CRC_VALUE;
  Where1  = ftell(F2);
  TotalWrit+=sizeof(Huffman); // everything that's in the archive m
ust be counted !
  if(fwrite(&Huffman, 1, sizeof(Huffman), F2) != sizeof(Huffman))
```

Trees

```
Error(3, "");

  Huffman._Block_Type_0 = FrozenBuffer(BitBufC,   BitBuf,   BLOCK_T
YPE_COMPRESSION);
  Huffman._Block_Type_1 = FrozenBuffer(LitBufC,   LitBuf,   BLOCK_T
YPE_COMPRESSION);
  Huffman._Block_Type_2 = FrozenBuffer(LenBufC,   LenBuf,   BLOCK_T
YPE_COMPRESSION);
  Huffman._Block_Type_3 = FrozenBuffer(DistBufC,  DistBuf,  BLOCK_T
YPE_COMPRESSION);
  Huffman._Block_Type_4 = FrozenBuffer(RestBufC2, RestBuf2, BLOCK_T
YPE_COMPRESSION);

  if(D_REST_BITS > 8)
  {
    for (i = 0; i < RestBufC1; i++)    send_bits(RestBuf1[i], (D_RE
ST_BITS - 8));
    updcrc((unsigned char *)RestBuf1, (unsigned int)RestBufC1 * 2,
&CRC);
  }
  Huffman._RestBits  = restbits + (end_of_block == 1 ? 9 : 0);
  Huffman._CRC_total = CRC;

  Where2 = ftell(F2);
  fseek(F2, Where1, SEEK_SET);
  if(fwrite(&Huffman, 1, sizeof(Huffman), F2) != sizeof(Huffman))
Error(3, "");
  fseek(F2, Where2, SEEK_SET);
  bits_flush();
  Tree_Start();
}

int Read_Block()
{ unsigned int i;
  unsigned short Value;

  int u= ftell(F1);
  // it should have been only for the debug procedures ...

  if(fread(&Huffman, 1, sizeof(Huffman), F1) != sizeof(Huffman))  E
rror(4, "");
  restbits = Huffman._RestBits;
  HuffmanBlockCRC  = Huffman._CRC_total;
  CRC = INIT_CRC_VALUE;

  MeltBuffer(&BitBufC,   BitBuf,   Huffman._Block_Type_0, BLOCK_TYP
```

Trees

```
E_COMPRESSION);
  MeltBuffer(&LitBufC,   LitBuf,   Huffman._Block_Type_1, BLOCK_TYP
E_COMPRESSION);
  MeltBuffer(&LenBufC,   LenBuf,   Huffman._Block_Type_2, BLOCK_TYP
E_COMPRESSION);
  MeltBuffer(&DistBufC,  DistBuf,  Huffman._Block_Type_3, BLOCK_TYP
E_COMPRESSION);
  MeltBuffer(&RestBufC2, RestBuf2, Huffman._Block_Type_4, BLOCK_TYP
E_COMPRESSION);
  RestBufC1 = RestBufC2 = DistBufC;

  u= ftell(F1);

  if(D_REST_BITS > 8)
  {
    GetW(bits_buf);
    bits_len = 16;
    for (i = 0; i < RestBufC1; i++)
    {
       Value = 0;
       get_bits(&Value, (D_REST_BITS - 8), bool(i < RestBufC1 - 1))
;
       RestBuf1[i] = Value;
    }
    updcrc((unsigned char *)RestBuf1, (unsigned int)RestBufC1 * 2,
&CRC);
  }
  else memset((unsigned char *)RestBuf1, 0, sizeof(RestBuf1));

  LitBufC  = LenBufC = DistBufC = RestBufC1 = RestBufC2 = DistBufC
= 0U;
  u= ftell(F1);

  if(CRC != HuffmanBlockCRC)
  {
    if(Testez == 0) Error(2, " ");
    TestedOk = 0;
    return 1;
  }
  if(Testez == 0) Printf(" Melted OK! ", "\r", 3);
  return 0;
}

int Tree_Encode(int distance, int len)
{
   restbits = ((restbits + 1) % 8);
```

Trees

```
   if(distance)
   {
     distance --;
     DistBuf [DistBufC++] =  (distance >> D_REST_BITS) & 0x00ff;
     RestBuf1[RestBufC1]  =  distance & B_MASK[D_REST_BITS];
     RestBuf2[RestBufC2]  =  (unsigned char)(RestBuf1[RestBufC1] >>
 (D_REST_BITS - 8));
     RestBuf1[RestBufC1] &=  B_MASK[(D_REST_BITS - 8)];
     // ex: D_REST_BITS=10 ==> D_REST_BITS - 8 = 2
     // ==> deplasam cu D_REST_BITS - 8 = 2 pentru a obtine primii
8 biti == alt char!
     RestBufC1++;  RestBufC2++;
     if(len < 255) LenBuf[LenBufC++]=len;
     else
      {  LenBuf[LenBufC++] = (unsigned char) (255); // marker
         LenBuf[LenBufC++] = (unsigned char) (len);
         LenBuf[LenBufC++] = (unsigned char) ((len >> 8) & 0x00ff);
    } }
   else
   {
     flag |= maska;
     LitBuf[LitBufC++] = len;
   }
   if ((maska <<= 1) == 0)  // deja 8 ?
     {
        BitBuf[BitBufC++] = flag;
        flag = 0;
        maska = 1;
     }
   if(LitBufC > LITBUF - 8 || LenBufC > LITBUF - 8 || DistBufC > LI
TBUF - 8 || RestBufC1 > LITBUF - 8 || BitBufC > LITBUF - 8)
   {
      Send_Block(0);
   }
   return 0;
}

void Tree_Flush()
{
  if(LitBufC || LenBufC || DistBufC) Send_Block(1);
  Solid_CRC = INIT_CRC_VALUE;
  updcrc((unsigned char *)Fereastra, (unsigned int)string_starting,
 &Solid_CRC);
}

void send_bits(unsigned short value, short length)  /* value to sen
```

Trees

```
d , number of bits */
 {
    if (bits_len > 16 - length) // let's try with 32 and a real put
w()...
       {
         bits_buf <<= 16 - bits_len;
         bits_buf  += value >> (length - (16 - bits_len));
         PutW(bits_buf);
         bits_buf = (unsigned short)(value & B_MASK[length - (16  -
bits_len)]);
         bits_len += length - 16;
       }
      else
       {
         bits_buf <<= length;  // the easiest way to send is when
         bits_buf  += value;   // the bits number lies within the
         bits_len  += length;  // 16-bit boundaries.
       }
 }
void get_bits(unsigned short *value, short length, bool ok)  /* val
ue where to get , number of bits, contor */
 {
    if (bits_len >= length)
      {
        *value = bits_buf >> (bits_len - length);
        if(bits_len - length == 0 && ok)
         {
            GetW(bits_buf);
            bits_len = 16;
         }
        else
        {
  //  ------------
            bits_buf = bits_buf & B_MASK[bits_len -length]; // 000 0
00 0 | 000 001 1 | 00
            bits_len -= length;
         }
       }
      else
       {
          *value = bits_buf;
          GetW(bits_buf);
          *value <<= length - bits_len;
          *value |= bits_buf >> (16 - (length - bits_len));
          bits_len = 16 - (length - bits_len);
          bits_buf = bits_buf & B_MASK[bits_len]; // 000 000 0 | 000
```

Trees

```
 001 1 | 00
       }
}

// Send the rest of the word, when the compression ends and there i
s
// no other way to fill it up.

void bits_flush(void)
{
  // strange but first this resolved a bug, then created others !!!
  // if(bits_len)  // don't waste 2 byte if there is nothing to put
 !
  //the same condition as with the other bits : if(bits_len) is not
 enough !

  if(D_REST_BITS > 8)
   {
     bits_buf <<=  (unsigned short) (16 - bits_len);          // f
ill it till
     PutW(bits_buf);
  }
}
```

Nonsolid

```
/*****************************************************************
***********
 *
          *
 *              Non solid stuff but still useful procedures
          *
 *
          *
 *          From Xtreme 106 (DLL) package by Sabin, Belu (c) 1999
          *
 *
          *
 *****************************************************************
***********/
#pragma pack (1)

#include <io.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <memory.h>
#include <string.h>

#include <dos.h>
#include <direct.h>

#include "util.h"
#include "trees.h"
#include "solid.h"
#include "markov.h"
#include "deflate.h"
#include "console.h"
#include "nonsolid.h"


int Write_Fake_Header(void)  // for non solid operations
 {char TmpFil[_MAX_PATH];

  header_pos  = ftell(F2);
  strcpy(TmpFil, (char *)&InFile[strstr(InFile, ":\\") ? 3 : 0]);

  XTREME_H._NamLen  = strlen(TmpFil);
  if(fwrite(&XTREME_H,  1, sizeof(XTREME_H), F2) != sizeof(XTREME_H
) || fwrite(TmpFil, 1, XTREME_H._NamLen, F2) != XTREME_H._NamLen)
  {
     Error(7, "");
```

Nonsolid

```
      return 1;
    }
    return 0;
  }

int Write_Real_Header(FILE *F)
 {
      rewind(F);
      fseek (F, header_pos, SEEK_SET);

      XTREME_H._SizeUnCom  = TotalSum;
      XTREME_H._Time       = 0;
      XTREME_H._Date       = 0;
      XTREME_H._FileCRC    = 0;
      XTREME_H._Table      = TableIndex;
      XTREME_H._NextFile = TotalWrit + sizeof(XTREME_H) + XTREME_H._
NamLen;
      fwrite(&XTREME_H, 1, sizeof(XTREME_H), F);
      fseek (F, 0L, SEEK_END);
      return 0;
}


void PackFile(char *name, unsigned int size)
{  char TmpFile[_MAX_PATH];
   unsigned long verif;

   memset(TmpFile, 0, _MAX_PATH);
   strcpy(TmpFile, name);

   strcpy(InFile,  TmpFile);
   if(Solida == 0) Write_Fake_Header();
   TotalRead = TotalWrit = 0;
   TotalSum = size;
   if(Repack == 0 || Solida == 0) ReadjustTables(); //TotalSum

   if((F1 = fopen(TmpFile, "rb" )) == NULL )  Error(0, TmpFile);
   else
   {
     verif = _filelength(fileno(F1));
     if(verif != size)  Error(10, "");    // file size changed sinc
 e scanning !!!!
     initialize_engine(5);
     deflate();
   }
   Fclose(F1, 0, 0);
```

Nonsolid

```
    if(Solida == 0)
    {
      sprintf(TmpFile, "%-60s" , short_str(InFile, 60));
      Print(Repack ? " Repacked" : " Freezed", " ", 3, TmpFile, "\n"
  , 2, "", "", 0, "", "", 0,  -1);
      Write_Real_Header(F2);
    }
}

void Freeze(char *FileMask, char repack)
{
  char tmp [_MAX_PATH];
  unsigned int i;

  SolidInit(); SolidN = 0;
  Register_File_List(FileMask);

  if(Repack == 0)
  {
    Printf(" Scanning complete. ", "\n", 3);
    sprintf(tmp, "%s '%s'", short_str(FileMask, 56), (Repack ? " Re
packing files " : " Freezing files "));
    Printf (tmp, "\n", 2);
  }

  if(SolidN)
  {
    if(repack == 0) Write_Archive_Header();
    for(i=0; i < SolidN; i++)
                PackFile(SolidArray[i].RealName, SolidArray[i].siz
e);
  }
}

void MeltFile()
{
  char FileN[_MAX_PATH];
  char tmp[_MAX_PATH];

  if(Read_Archive_Header(1)) { Error(1, ""); return; }
  TotalRead = TotalWrit = 0;
  memset(FileN, 0, _MAX_PATH);
  AssumeYes = 0; // once given, does not mean for all the archives,
 nu ?
  if(Solida == 0)
  {
```

Nonsolid

```
    while(!CheckStruct(FileN))
    {
      if(FileN[0])
      {
        if(OpenFile(FileN, 0, 32)) // normal attribute
          {
            memset(InFile, 0, _MAX_PATH);
            strcpy(InFile, FileN);
            inflate();
            Fclose(F2, 0, 0);
          }
      }
     memset(FileN, 0, _MAX_PATH);
    }
  }
  else
  {
          int uu = ftell(F1);
      SolidMelt();
      sprintf(tmp, "%d files processed.", SolidK);
      Printf(tmp, "\n", 10);
  }
}

int TestFrostedFiles()
{ char FileN[_MAX_PATH];
  unsigned long CatAre = _filelength(_fileno(F1));
  unsigned long Undere = 0, Errors = 0;

  if(Read_Archive_Header(1) || CatAre == 0) { Error(1, ""); exit(0)
; }
  memset(FileN, 0, _MAX_PATH);
  if(Solida == 0)
   {
     while(!CheckStruct(FileN))
      {
       TestedOk = 1;
       if(FileN[0])
        {
          Printf(" Testing", " " ,3);
                  Printf(FileN, "\n", 2);

          for(;Read_Block() == 0;)
          {
             Printf(" Testing", " ", 3);
                        Printf(FileN, "\n ", 2);
```

Nonsolid

```
            if(restbits > 8) break;
          }
          sprintf(tmp, " %-70s", FileN);
          if(TestedOk)
                  {
                          Printf(tmp, " ", 2);
                          Printf(" OK", "\n", 3);
                  }
          else
          {
              Printf(tmp, " ", 2);
                          Printf(" failed", " \n", 12);
              Errors++;
          }
        }
        memset(FileN, 0, _MAX_PATH);
      }
    }
  else Errors = TestSolidArchive();

  return (Errors + 7);
}

int DeleteNonSolidArchive(char *files)
{
  char files1[_MAX_PATH], dir1[_MAX_PATH], ext1[_MAX_PATH];
  char files2[_MAX_PATH], dir2[_MAX_PATH], ext2[_MAX_PATH];
  char temp[_MAX_PATH],   tempfile[_MAX_PATH];
  char FileN[_MAX_PATH];
  unsigned int  c, dels = 0, k;

  memset(FileN, 0, _MAX_PATH);
  GetTemporaryDirectory(tempfile);
  strcat(tempfile, "xtremly.tmp");
  if((F2=fopen(tempfile, "wb+")) == NULL) return 1;

  Printf(" Deleting ", "'", 3); Printf(files, "\n", 2);
  Printf(" Creating temporary archive ", "'", 3); Printf(tempfile,
"\n", 2);
  Split(files, NULL, dir1, files1, ext1);
  Printf(" Scanning for packed ", "'", 3);  Printf(files, "\n", 2);

  if(Read_Archive_Header(1)) { Error(1, ""); exit(0); }
  else
  {
     //if((fwrite(&ARH_H, 1, sizeof(ARH_H), F2)) != sizeof(ARH_H))
```

Nonsolid

```
return 1; // writes the first header
     if(FWrite_ARH_Header(&ARH_H, F2)) return 1; // rewritten, has
a 4 byte CRC built in !
   }
   while(!CheckStruct(FileN))
   {
    if(FileN[0])
     {
       Split(FileN, NULL, dir2, files2, ext2);
       if(Verify(files2, files1, strlen(files2), strlen(files1)) &&
          Verify(ext2,     ext1, strlen( ext2), strlen( ext1)) &&
          ((strstr(dir2, dir1) == dir2 && (Recursiv    )) ||
           (stricmp(dir2, dir1) == 0    && (Recursiv == 0))))
       {
         Printf(" Deleted ", " ", 2);
                Printf(FileN, "\n", 3);  // just erase the file by
skiping it
         SkipToNextFile(F1);
         dels++;
       }
       else
       {
         if((fwrite(&XTREME_H, 1, sizeof(XTREME_H), F2)) != sizeof(X
TREME_H)) return 1;
         if((fwrite(FileN,     1, XTREME_H._NamLen, F2)) != XTREME_H
._NamLen) return 1;
         c = ftell(F1);
         for(;;)
         {
           if((k = getc(F1)) == EOF) Error(9, ""); // unexpected END
-OF-Archive...
           putc(k, F2);
           if(c++ >= (header_pos + sizeof(ARH_H)) - 1) break;
         }
         c = ftell(F1);
       }
     } memset(FileN, 0, _MAX_PATH);
   }
  Fclose(F1, 0, 0);
  Fclose(F2, 0, 0);

  sprintf(temp, " %d file(s) deleted.", dels);
  Printf (temp, "\n", 2 );
  Printf (" Deleting temporary files ", " ", 2);

  if(dels)
```

Nonsolid

```
{
  //GetCurDir(temp);
  //strcat(temp, Destination);

  GetCurDir(temp);
  if(strstr(Destination, ":\\") == NULL) strcat(temp, Destination)
;
  else strcpy(temp, Destination);

 _unlink(temp);
  rename(tempfile, temp);
  return 0; // archive has just been changed !
 }
 else
 {
  _unlink(tempfile); // nothing to replace
  return 1;
 }
 return -1;
}

void NonSolidListFiles(char *files)
{ char files1[_MAX_PATH], dir1[_MAX_PATH], ext1[_MAX_PATH];
  char files2[_MAX_PATH], dir2[_MAX_PATH], ext2[_MAX_PATH];

  char FileN[_MAX_PATH];
  unsigned int FileK=0;

  memset(FileN, 0, _MAX_PATH);
  sprintf(tmp, " Name %55s %10s %s", "Original", "Compressed", "Rat
io");
  sprintf(tmp2, " ----------------------------------------------
----------------------------");

  Printf(tmp,  "\n",  3);
  Printf(tmp2, "\n", 15);

  Split(files[0] == 0 ? "*.*" : files, NULL, dir1, files1, ext1);

  header_pos = 0;

  while(!CheckStruct(FileN))
    {
      if(FileN[0])
      {
        Split(FileN, NULL, dir2, files2, ext2);
```

Nonsolid

```
        if( Verify(files2, files1, strlen(files2), strlen(files1))
&&
            Verify(ext2,     ext1, strlen(  ext2), strlen(  ext1))
&&
            ((strstr(dir2, dir1) == dir2 && (Recursiv     )) ||
             (stricmp(dir2, dir1) == 0    && (Recursiv == 0)) || fi
les[0] == 0))
        {
           ratio = IntoarceProcent(TotalWrit, TotalRead);
           sprintf(tmp,  " %-48s", short_str(FileN, 48));
           sprintf(tmp2, "%10d %10d  %3d.%d%c", TotalRead, TotalWri
t, ratio / 10, ratio % 10, '%');
           Print  (" ", " ", 3, tmp, " ", 2, tmp2, "\n", 3, " ", "
", 0, -1);
           FileK++;
        }
     }
    memset(FileN, 0, sizeof(FileN));
    SkipToNextFile(F1);
   }

   sprintf(tmp2," ----------------------------------------------------
---------------------------");
   sprintf(tmp, " %d file(s).", FileK);
   Printf(tmp2, "\n", 15);
   Printf(tmp,  "\n", 10);
}

int UpdateNonSolidArchive(char *files, char _to_what_type)
{
   char files1[_MAX_PATH], dir1[_MAX_PATH], ext1[_MAX_PATH];
   char files2[_MAX_PATH], dir2[_MAX_PATH], ext2[_MAX_PATH];
   char temp[_MAX_PATH],   tempfile[_MAX_PATH];
   char FileN[_MAX_PATH];
   unsigned int  c, dels = 0, k;

   memset(FileN, 0, _MAX_PATH);
   GetTemporaryDirectory(tempfile);
   strcat(tempfile, "xtremly.tmp");
   if((F2=fopen(tempfile, "wb+")) == NULL) return 1;

   Printf(" Deleting ", "'", 3); Printf(files, "\n", 2);
   Printf(" Creating temporary archive " ,"'", 3);  Printf(tempfile,
 " ", 2);
   ComplexSplit(files, temp, dir1, files1, ext1);
```

Nonsolid

```
  if(Read_Archive_Header(1)) { Error(1, ""); return 1; }
  else
  {
     //if((fwrite(&ARH_H, 1, sizeof(ARH_H), F2)) != sizeof(ARH_H))
return 1; // writes the first header
     if(FWrite_ARH_Header(&ARH_H, F2)) return 1; // rewritten, has
a 4 byte CRC built in!
  }
  Printf(" Scanning for packed ", "'", 3); Printf(files, "\n", 2);

  while(!CheckStruct(FileN))
  {
   if(FileN[0])
    {
      Split(FileN, NULL, dir2, files2, ext2);
      if(Verify(files2, files1, strlen(files2), strlen(files1)) &&
         Verify(ext2,     ext1, strlen(  ext2), strlen(  ext1)) &&
         ((strstr(dir2, dir1) == dir2 && (Recursiv     )) ||
         (stricmp(dir2, dir1) == 0    && (Recursiv == 0))))
      {
        Printf(" Deleted ", " ", 2); Printf(FileN, "\n", 3);// just
 erase the file by skiping it
        SkipToNextFile(F1);
        dels++;
      }
      else
      {
        if((fwrite(&XTREME_H, 1, sizeof(XTREME_H), F2)) != sizeof(X
TREME_H))  return 1;
        if((fwrite(FileN,     1, XTREME_H._NamLen, F2)) != XTREME_H
._NamLen)  return 1;
        c = ftell(F1);
        for(;;)
        {
          if((k = getc(F1)) == EOF) Error(9, ""); // unexpected END
-OF-Archive...
          putc(k, F2);
          if(c++ >= (header_pos + sizeof(ARH_H)) - 1) break;
        }
        c = ftell(F1);
      }
    } memset(FileN, 0, _MAX_PATH);
  }
 sprintf(temp, " %d  files will be replaced.", dels);

 Printf (temp, "\n", 2);
```

Nonsolid

```
 Fclose(F1, 0, 0); Repack = 1;
 Freeze(files, 1); // repacking
 Fclose(F1, 0, 0);
 Fclose(F2, 0, 0);

 Printf(" Deleting temporary files", " ",  2);

 //if(dels) nu trebuie neaparat sa fie ceva de sters
 //GetCurDir(temp);
 //strcat(temp, Destination);

 GetCurDir(temp);
 if(strstr(Destination, ":\\") == NULL) strcat(temp, Destination);
 else strcpy(temp, Destination);

_unlink(temp);
 rename(tempfile, temp);
 return 0;              // everything went okay !
}


/*
Analyzing archive depth
Highest revision number is 0
Saving archive tags
Creating empty target archive
Updated archive is empty
Damage protecting archive
Processing revision free files
Analyzing
Decompressing CHALES\LZCB_XXX\LZCB_XXX.NCB  ERROR 90: file is damag
ed
Decompressing CHALES\LZCB_XXX\RESTBITS OK
Decompressing CHALES\RAX2\RAX2.MAK OK
Decompressing CHALES\RAX2\DEFLA32.CP OK
Decompressing CHALES\RAXX\RES\RAXX.RC2 OK
Decompressing CHALES\RAXX\RES\ICONLIST.ICO OK
Decompressing CHALES\RAXX\RES\ICON4.ICO OK
Decompressing CHALES\RAXX\RES\ICON3.ICO OK
Decompressing CHALES\RAXX\RES\ICON2.ICO OK
Decompressing CHALES\RAXX\RES\CTRLDEMO.ICO OK
Decompressing CHALES\RAXX\RES\ICON1.ICO OK
Decompressing CHALES\RAXX\RES\RAXXDOC.ICO OK
Decompressing CHALES\RAXX\RES\RAXX.ICO OK
Decompressing CHALES\RAXX\RES\RAXZ.GIF OK
```

Nonsolid

```
Decompressing CHALES\RAXX\RES\CURSOR1.CUR OK
Decompressing CHALES\RAXX\RES\MM1.AVI 14%


*/
```

Markov

```
/*****************************************************************
***********
 *
          *
 *              Dynamic Markov Encoding/Decoding procedures
          *
 *
          *
 *          From Xtreme 106 (DLL) package by Sabin, Belu (c) 1999
          *
 *
          *
 *                        Fully 32 bit version !
          *
 *              Modified to be used in the XTREME project !
          *
 *                    ( and OPTIMIZED as much as posible )
          *
 *
          *
 *****************************************************************
***********/

#pragma pack (1)

#include  <io.h>
#include  <conio.h>
#include  <stdio.h>
#include  <string.h>
#include  <stdlib.h>
#include  <process.h>

#include  "markov.h"
#include  "util.h"
#define    MarKov 256

long            MaximumSize;                    // i need a signed here
 !
short           Current_State=0;
struct    Tree  *STree[ MarKov ];
unsigned char   Bit[8] = {1,2,4,8,16,32,64,128};
unsigned char   Pack   = 0, Power  = 0;

short     DeNode=1, c;
long      R=0, W=0, S=0, FSize;
int       LastCh;
```

Markov

```
void De_Init_Buf(void)
{
  struct Tree *Tmp;

  for (short y=0; y<MarKov; y++) {    Tmp=STree[ y ];  free (Tmp);
  }
}

void InitializeMarkovStates(void)
{ unsigned short  s;

  for(s=0; s < MarKov; s++)
   {
     if((STree[s] = (struct Tree *)malloc(sizeof(struct Tree)))==NU
LL) abort();
   }
}

void InitModel(void)
{  unsigned short e, s;
   Pack = 0; Power  = 0; W = 0; Current_State = 0;

   for( s=0; s < MarKov; s++)
   {
     for ( e=2; e < 512; e++)   STree[s]->Spl_Dad[e] = e >> 1 ;
     for ( e=1; e < 256; e++)
      {
         STree[s]->Spl_St[e]   = e+e;
         STree[s]->Spl_Dr[e]   = STree[s]->Spl_St[e]+1;
      }
   }
}

void SemiSplay(unsigned char C)
{ short Dad1, Dad2, Aunt, Node;

  struct Tree *Tmp;

  Tmp=STree[Current_State];
  Node = C + 256;
  do
  {
    Dad1     = Tmp->Spl_Dad [ Node ];
    if (Dad1 != 1 )
```

Markov

```
    {
      Dad2   = Tmp->Spl_Dad [ Dad1 ];
     if(Dad1 ==Tmp->Spl_Dr  [ Dad2 ])
                {
                  Aunt = Tmp->Spl_St [ Dad2 ];
                  Tmp->Spl_St [ Dad2 ] = Node;
                 }
           else {
                  Aunt = Tmp->Spl_Dr [ Dad2 ];
                  Tmp->Spl_Dr [ Dad2 ] = Node;
                }
      if ( Node == Tmp->Spl_St [ Dad1 ] )  Tmp->Spl_St [ Dad1 ] = A
unt;
            else                           Tmp->Spl_Dr [ Dad1 ] = A
unt;

       // Exchange Dads

        Tmp->Spl_Dad [ Aunt ] = Dad1;
        Tmp->Spl_Dad [ Node ] = Dad2;

       Node = Dad2;
     }
      else Node  = Dad1;

  } while( Node != 1 );

  if (MarKov != 1) Current_State = C % MarKov;       // MarKov Curre
nt_State Change
}


/*****************************************************************
*****************
                These are Markov Encoding and Decoding functions
 *****************************************************************
*****************/

void MarkovEncode(unsigned int Size, unsigned char *Buffer, unsigne
d char *OutBuff, unsigned int *outcnt)
{ register short  Node1, Node2;
  unsigned char   Stack[255], T=0;
  register struct Tree *Tmp;
  unsigned short  s,e;
  unsigned char   C;
```

Markov

```
Pack = 0; Power  = 0; W = 0; Current_State = 0;
for(s=0; s < MarKov; s++)
 {
   Tmp=STree[s];
   for ( e=2; e < 512; e++)    Tmp->Spl_Dad[e] = e >> 1 ;
   for ( e=1; e < 256; e++)
   {
      Tmp->Spl_St[e]   = e+e;
      Tmp->Spl_Dr[e]   = Tmp->Spl_St[e]+1;
   }
 }

short Dad1, Dad2, Aunt, Node;

for(unsigned int i=0; i<Size; i++)
{
   C = Buffer[i];

   Tmp    = STree[Current_State];
   Node1 = C + 256;
   do
   {
     Node2       =  Tmp->Spl_Dad [ Node1 ];
     Stack[++T] = (Tmp->Spl_Dr   [ Node2 ] == Node1) ;
     Node1       =  Node2;

   }while(Node1 != 1);

   do{
       if(Stack[T--]) Pack+=Bit[Power];
       if(++Power == 8)
       {
            Power=0;
            OutBuff[W++] = Pack;
            Pack=0;
       }
   } while(T >= 1 );

   Node = C + 256;
   do
   {
     Dad1      = Tmp->Spl_Dad [ Node ];
     if (Dad1 != 1 )
     {
        Dad2    = Tmp->Spl_Dad [ Dad1 ];
        if(Dad1 ==Tmp->Spl_Dr   [ Dad2 ])
```

Markov

```
            {
                Aunt = Tmp->Spl_St [ Dad2 ];
                Tmp->Spl_St [ Dad2 ] = Node;
            }
          else
          {
                Aunt = Tmp->Spl_Dr [ Dad2 ];
                Tmp->Spl_Dr [ Dad2 ] = Node;
          }
          if ( Node == Tmp->Spl_St [ Dad1 ] )  Tmp->Spl_St [ Dad1 ]
= Aunt;
          else                                 Tmp->Spl_Dr [ Dad1 ]
= Aunt;

          // Exchange Dads

          Tmp->Spl_Dad [ Aunt ] = Dad1;
          Tmp->Spl_Dad [ Node ] = Dad2;

        Node = Dad2;
      }
    else Node  = Dad1;

   } while( Node != 1 );
   if (MarKov != 1) Current_State = C % MarKov;       // MarKov Curr
ent_State Change
  }

  if(Power) OutBuff[W++] = Pack;
  *outcnt = W;
}

void MarkovDecode(unsigned int Size, unsigned char *Buffer, unsigne
d char *OutBuff, unsigned int *outcnt)
{
  unsigned short  s,e;
  struct   Tree  *Tmp;
  unsigned char   Ch;

  W = Current_State = 0;
 for( s=0; s < MarKov; s++)
  {
    Tmp=STree[s];
    for ( e=2; e < 512; e++)   Tmp->Spl_Dad[e] = e >> 1 ;
    for ( e=1; e < 256; e++)
     {
```

Markov

```
            Tmp->Spl_St[e]   = e+e;
            Tmp->Spl_Dr[e]   = Tmp->Spl_St[e]+1;
        }
      }

    DeNode = 1;

    for(unsigned int i=0; i<Size; i++)
    {
      Ch = Buffer[i];
      Tmp=STree[Current_State];

      for(short w=0; w < 8; w++)
      {
        DeNode=(( Ch % 2 == 0 ) ? Tmp->Spl_St [ DeNode ] : Tmp->Spl_D
r [ DeNode ]);
        if (DeNode >= 256)
          {
             SemiSplay (DeNode-256);
             OutBuff[W++] = DeNode-256;
             DeNode=1;                          // Get THE   ROOT
             Tmp=STree[Current_State];          // On  == Another == Tree
        }
        Ch >>= 1;
      }
      LastCh = Ch;
    }
    *outcnt = W;
}
```

Mainu

```
/*****************************************************************
***********
 *
          *
 *
                  Xtreme 106 DLL Main file / procedures
          *
 *
          *
 *          From Xtreme 106 (DLL) package by Sabin, Belu (c) 1999
          *
 *
          *
 *****************************************************************
***********/

#pragma pack (1)

#include <io.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <memory.h>
#include <string.h>

#include <dos.h>
#include <direct.h>

#include "util.h"
#include "trees.h"
#include "solid.h"
#include "markov.h"
#include "deflate.h"
#include "console.h"
#include "recovery.h"
#include "nonsolid.h"

#include "exports.h"

FILE *F1, *F2;

__declspec( dllexport ) void DeInit_Xtreme106_Engine(void);

#pragma comment( exestr, "Xtreme 106 DLL compiled by Sabin Belu on
" __DATE__ " at " __TIME__ )
```

Mainu

```
#pragma comment( exestr, "Xtreme 106 DLL authenticity verification
ID no. : 0x47C1E0020305F403")

struct Shir *Sursa;
char extractionDestination[_MAX_PATH];
char Destination[_MAX_PATH];
char Inghetz = 0, Testez = 0, Sterg = 0, Extrag = 0, TestedOk = 1,
Listez = 1,
     Actualizez = 0, Repack = 0, Protejez = 0, Repar = 0, ZipCloaki
ng = 0,
     Mut = 0, Incui = 0;

unsigned char  *Fereastra = NULL;
struct NewInteger   *prev = NULL;
struct NewInteger   *head = NULL;
unsigned char   *l_buf    = NULL;
unsigned short  *d_buf    = NULL;
unsigned char   *BitBuf   = NULL; unsigned int BitBufC;
unsigned char   *LenBuf   = NULL; unsigned int LenBufC;
unsigned char   *LitBuf   = NULL; unsigned int LitBufC;
unsigned char   *DistBuf  = NULL; unsigned int DistBufC;
unsigned char   *RestBuf2 = NULL; unsigned int RestBufC2;
unsigned short  *RestBuf1 = NULL; unsigned int RestBufC1;

unsigned int  XXXInBufC, XXXOutBufC;
unsigned char *XXXInBuf, *XXXOutBuf;

char InFile[_MAX_PATH];
long TotalRead, TotalWrit, TotalSum;
char TableIndex = 1, Solida = 0,
     Recursiv   = 0, AssumeYes = 0, archive_type;
extern struct TableData configuration_table[6];


int PreEmptiveInit(void)
{
  // very important .... since we are doing a CRC on this structure
 ....
  memset((unsigned char *)&ARH_H, 0, sizeof(ARH_H));
  return 0;
}

int PostEmptiveInit(void)
{
  unsigned long MemUsed = 0, Mem1, Mem2;
  unsigned char Uol[30];
```

Mainu

```
    ratio = 0;

    if(Fereastra == NULL)
    {
      MemUsed+= 2L*WSIZE + 6*LITBUF_TO_ALLOC;
      if(Inghetz || Sterg) MemUsed+= sizeof(struct NewInteger)*WSIZE
 + sizeof(struct NewInteger)*HASH_SIZE;
      Mem1 =  MemUsed / (1024*1024);  Mem2 = ((MemUsed - Mem1*(1024*1
024))/1000) % 1000;
      sprintf((char *)Uol, "%ld.%ld", Mem1, Mem2);

      if(Repack == 0)
          {
        Print(" Requesting", " ", 3, (char *)Uol, " ", 2, "Mb.", "",
3, "", "", 0, -1);
          }

      if(ZipCloaking) Printf(" Initializing Zip Cloaking ...", " ", 1
4);

      memory_request_alloc(unsigned char,Fereastra,2L*WSIZE);
      if(Inghetz || Sterg)
      {
        memory_request_alloc(struct NewInteger,prev,WSIZE);
        memory_request_alloc(struct NewInteger,head,HASH_SIZE);
      }
      memory_request_alloc(unsigned char,  BitBuf,     LITBUF_TO_ALLOC
);
      memory_request_alloc(unsigned char,  LenBuf,     LITBUF_TO_ALLOC
);
      memory_request_alloc(unsigned char,  LitBuf,     LITBUF_TO_ALLOC
);
      memory_request_alloc(unsigned char,  DistBuf,    LITBUF_TO_ALLOC
);
      memory_request_alloc(unsigned char,  RestBuf2,   LITBUF_TO_ALLOC
);
      memory_request_alloc(unsigned short, RestBuf1,   LITBUF_TO_ALLOC
);
      memory_request_alloc(unsigned char,  XXXInBuf,   LITBUF_TO_ALLOC
);
      memory_request_alloc(unsigned char,  XXXOutBuf, LITBUF_TO_ALLOC
);
      if(Fereastra == NULL || ((prev == NULL || head == NULL) && Ingh
etz))  { Error(11, ""); exit(0); }
```

Mainu

```
    memset(Fereastra, 0, 2L*WSIZE);
  }
  return 0;
}

void DeInit_Xtreme106_Engine()
{
  memory_request_free(Fereastra);
  memory_request_free(prev);
  memory_request_free(head);
  memory_request_free(BitBuf);
  memory_request_free(LenBuf);
  memory_request_free(LitBuf);
  memory_request_free(DistBuf);
  memory_request_free(RestBuf1);
  memory_request_free(RestBuf2);
}

void ReadjustTables(void)
{
  WSIZE          = configuration_table[TableIndex].WSIZE_INDEX*0x800
0;
  TOO_DISTANT    = configuration_table[TableIndex].TOO_DISTANT*2096;
  window_size    = (ulg)2*WSIZE;
  MIN_MATCH      = configuration_table[TableIndex].MIN_MATCH;//3;
  MAX_MATCH      = configuration_table[TableIndex].MAX_MATCH;
  MIN_LOOKAHEAD  = (MAX_MATCH+MIN_MATCH+1);
  HASH_BITS      = configuration_table[TableIndex].HASH_BITS;
  HASH_SIZE      = (unsigned)(1<<HASH_BITS);
  HASH_MASK      = (HASH_SIZE-1);
  WMASK          = (WSIZE-1);
  MAX_DIST       = (WSIZE-MIN_LOOKAHEAD);
  H_SHIFT        = ((HASH_BITS+MIN_MATCH-1)/MIN_MATCH);
  D_REST_BITS    = configuration_table[TableIndex].D_REST_BITS;
  PostEmptiveInit();
}

int DeleteFiles(char *files)
{
  if(Read_Archive_Header(1)) { Error(1, ""); exit(0); }
  if(Solida) return SolidDeleteFiles(files);
  else       return DeleteNonSolidArchive(files);
  return -1;
}

void ListFrozenFiles(char *files)
```

Mainu

```
{
 if(Read_Archive_Header(1))   { Error(1, ""); exit(0); }
 if(Solida == 0) NonSolidListFiles(files);
 else            SolidListFiles(files);
}

void DamageProtectArchive(char archivetype, char force_mode)
{
  if(Read_Archive_Header(0)) { Error(1, ""); return; }
  if(ARH_H._recovery_Filesize == 0 || force_mode)  DamageProtectorA
rchive();
  else
   Error(15, " "); //" Recovery record found !", " Archive is alrea
dy protected !");
}

void RemoveDamageProtection(char archivetype)
{
  if(ARH_H._recovery_Filesize)
  {
    WipeCorrectionTables(F1);
  }
  else
  {
    Error(14, " ");// " Recovery record not found !", " The archive
 must have been protected first ('p'rotect command) !");
  }
}

void ReAddProtection()
{
  Fclose(F1, 0, 0);
  Fclose(F2, 0, 0);
  metoda                      = ARH_H._recovery_Method;
  COMPRESS_RECOVERY_RECORD = ARH_H._recovery_Compressed;

  if((F1 = fopen(Destination, "rb+" )) == NULL ) Error(0, Destinati
on);
  Printf(" Readding data recovery records.", " ", 3);
  DamageProtectArchive(archive_type, 1);
}

void RepairDamagedArchive(char archivetype)
{
  if(Read_Archive_Header(0)) { Error(1, ""); return; }
  if(ARH_H._recovery_Filesize)
```

Mainu

```
   {
     if((F2 = fopen("repaired.xtr", "rb")) != NULL)
      {
         char t = ScanForAchar(" File repaired.xtr already exists! O
verwrite ? (Y)es,(E)nter/(N)o/(C)ancel... ", 4);
         switch(t)
          {
             case 'Y' : case 'y' : case 'E' : case 'e' : case 13 : F
close(F2, 0, 0); goto Rep; break;
          }
         goto Afar;
       }
      Rep:   if((F2 = fopen("repaired.xtr", "wb+")) == NULL) return;
             RepairArchive();
      Afar:;
   }
   else
   {
     Error(14, " ");// " Recovery record not found !", " The archive
 must have been protected first ('p'rotect command) !");
   }
}

void TryToRepair()
{
   if(ARH_H._recovery_Filesize == 0)  return;
   Fclose(F1, 0, 0); Fclose(F2, 0, 0);
   if((F1 = fopen(Destination,   "rb+" )) == NULL ) Error(0, Destin
ation);

   Printf(" Trying to repair archive ... ", " ",  3);
   Printf(" Creating repaired version, repaired.xtr ", " ", 11);
   RepairDamagedArchive(archive_type);
}

void LockArchive()
{
   if((fread(&ARH_H, 1, sizeof(ARH_H), F1)) !=  sizeof(ARH_H))  Err
or(1, ""); // error openning archive
   ARH_H._Locked = 1;
   rewind(F1);

   //if((fwrite(&ARH_H, 1, sizeof(ARH_H), F1)) !=  sizeof(ARH_H)) E
rror(7, ""); // error writting the archive header
   if(FWrite_ARH_Header(&ARH_H, F1)) Error(7, ""); // rewritten, ha
s a 4 byte CRC built in !
```

Mainu

```
    Printf(" Locked prefectly.", " ", 2);
}

void PrintLOGO()
{
  Printf(" ", "\n\n",  3);
  Printf(" XTREME DLL", " ", 16*0 + 10);
  Printf("Ver 1.06 The UltiMate Data Compression Tool. ", "\n\n",
3);
  Printf(" ** THIS is a SHAREWARE Data Compression Program, written
 by Belu Sabin.", "\n", 7);
  Printf(" ** It is not a REGISTERED program so, PLEASE, DO NOT use
 it ", "\n", 7);
  Printf(" ** in a business or anywhere else, except for EVALUATION
 PURPOSES.", "\n\n", 7);
}

void PrintLogo(char action, char archive_type, char _updated_type)
{
  sprintf(tmp,  " Using %s table size.", configuration_table[TableI
ndex].Descriptor);
  Printf (tmp, "\n", 2);

  switch(action)
  {
   case 0 : sprintf(tmp2, " Creating%sarchive '%s'",  archive_type
? " solid " : " ", short_str(Destination, 50));
            Printf(tmp2, "\n\n", 2);
            break;

   case 1 : sprintf(tmp2, " Extracting%sarchive '%s'",  archive_typ
e ? " solid " : " ", short_str(Destination, 50));
            Printf(tmp2, "\n\n", 2);
            break;

   case 2 : sprintf(tmp2, " Testing%sarchive '%s'",  archive_type ?
 " solid " : " ", short_str(Destination, 50));
            Printf(tmp2, "\n\n", 2);
            break;

   case 3 : sprintf(tmp2, " Listing%sarchive content '%s'",  archiv
e_type ? " solid " : " ", short_str(Destination, 50));
            Printf(tmp2, "\n\n", 2);
            break;
```

Mainu

```
   case 4 : sprintf(tmp2, " Deleting from%sarchive '%s'",  archive_
type ? " solid " : " ", short_str(Destination, 50));
            Printf(tmp2, "\n\n", 2);
            break;

   case 5 : sprintf(tmp2, " Updating%sarchive '%s'",  archive_type
? " solid " : " ", short_str(Destination, 50));
            Printf (tmp2,"\n\n", 2);
            break;

  case 6 : sprintf(tmp2, " Protecting%sarchive '%s' (adding ecc)",
  archive_type ? " solid " : " ", short_str(Destination, 50));
            Printf(tmp2, "\n\n", 2);
            break;

   case 7 : sprintf(tmp2, " Removing protection from%sarchive '%s'"
,  archive_type ? " solid " : " ", short_str(Destination, 50));
            Printf(tmp2, "\n\n", 2);
            break;

   case 8 :
            sprintf(tmp2, " Repairing%sarchive '%s'",  archive_type
 ? " solid " : " ", short_str(Destination, 50));
            Printf(tmp2, "\n\n", 2);
            break;

   case 9 : sprintf(tmp2, " Locking%sarchive '%s'",  archive_type ?
 " solid " : " ", short_str(Destination, 50));
            Printf(tmp2, "\n\n", 2);
            break;

  }
}


void CheckForMove()
{
  int deleted_files = 0;
  char tmp[_MAX_PATH];

  if(Mut)
  {
     Printf(" Deleting freezed files ", " ", 2);
     for(unsigned int i=0; i < SolidN; i++)
     {
         Printf(" Deleted ", " ", 3);
```

Mainu

```
                Printf(SolidArray[i].RealName, "\n " , 2);
          if(_unlink_path(SolidArray[i].RealName, 1) == 0) deleted_f
iles++;
      }
      sprintf(tmp, " %d moved to archive.", deleted_files);
      Printf (tmp, "\n", 10);
   }
}


void dll_main(void)
{char rez = 0;

 ratio = 0;
 PrintLOGO();
 Print_Start();
 PreEmptiveInit();

 // this line is more important then your life ! Chec it!
 if(sizeof(struct NewInteger) != 3) return;  // you're doomed forev
er !


 if(Listez)
    {
      if((F1 = fopen(Destination, "rb" )) == NULL ) Error(0, Destina
tion);
      GetArchiveType(&archive_type);
      PrintLogo(3, archive_type, 0);
      ListFrozenFiles(Sursa[0].Name);
   }
 else
 if(Inghetz)
    {
      if(MyXtrStructure.sfx || ZipCloaking || (F1 = fopen(Destinatio
n, "rb" )) == NULL )
      {
         if((F2  = fopen(Destination, "wb+" )) == NULL ) Error(0, De
stination);
         Actualizez = 0;
         PrintLogo(0, Solida, 0);

         if(Solida == 0) Freeze(Sursa[0].Name, 0);
          else           SolidFreeze(Sursa[0].Name);
      }
      else
```

Mainu

```
      {
         AssumeYes = 1;
         GetArchiveType(&archive_type);
         Actualizez = Repack = 1;
         PrintLogo(5, archive_type, Solida);

                 //if(Actualizez == 1) FileSelection();

         if(ARH_H._Locked)  Error(13, "");
         if(archive_type == 1)  rez = UpdateSolidArchive(Sursa[0].Na
me, Solida);  // to Solid, or to normal
         else                   rez = UpdateNonSolidArchive(Sursa[0]
.Name, Solida); // Solida is from input, so what the users requests
 for the next archive to be like !

         if(ARH_H._recovery_Filesize && rez == 0) ReAddProtection();
      }
      CheckForMove();
    }
  else
  if(Testez || Extrag)
    {
      if((F1 = fopen(Destination, "rb" )) == NULL ) Error(0, Destin
ation);
      GetArchiveType(&archive_type);
      PrintLogo(1+Testez, archive_type, 0);
      if(Extrag)  MeltFile();
      else
      {
        rez = TestFrostedFiles();
        TestRecoveryRecord();
        if(rez > 7) TryToRepair();
      }
    }
  else
  if(Sterg)
     {
       Repack = 1;
       AssumeYes = 1;
       if((F1 = fopen(Destination, "rb" )) == NULL ) Error(0, Destin
ation);
       GetArchiveType(&archive_type);
       PrintLogo(4, archive_type, 0);

     if(ARH_H._Locked)  Error(13, "");
       if(Sursa[0].Name) rez = DeleteFiles(Sursa[0].Name);
```

Mainu

```
      if(ARH_H._recovery_Filesize && rez == 0) ReAddProtection();
    }
 else
 if(Protejez)
    {
      if((F1 = fopen(Destination, "rb+" )) == NULL ) Error(0, Desti
nation);
      GetArchiveType(&archive_type);
      PrintLogo(6 + (Protejez - 1), archive_type, 0);
      if(ARH_H._Locked)  Error(13, "");

      if(Protejez == 1) DamageProtectArchive(archive_type, 0);
       else             RemoveDamageProtection(archive_type);
     }
 else
 if(Repar)
    {
      if((F1 = fopen(Destination,   "rb+" )) == NULL ) Error(0, Des
tination);
      GetArchiveType(&archive_type);
      PrintLogo(8, archive_type, 0);
      RepairDamagedArchive(archive_type);
 }
 else
 if(Incui)
 {
      if((F1 = fopen(Destination,   "rb+" )) == NULL ) Error(0, Des
tination);
      GetArchiveType(&archive_type);
      PrintLogo(9, archive_type, 0);
      if(ARH_H._Locked)  Error(13, "");
      LockArchive();
 }

 Fclose(F1,   0, 0);
 Fclose(F2,   0, 0);
 if(MyXtrStructure.EndDLLJob != NULL)  MyXtrStructure.EndDLLJob();
}
```

InitDLL

```
#pragma pack (1)

#include <io.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <memory.h>
#include <string.h>

#include <dos.h>
#include <direct.h>

#include "util.h"
#include "exports.h"

extern void dll_main(void);

__declspec( dllexport ) void FreeOtherStuff( void);
__declspec( dllexport ) int  Init_Xtreme106_Engine ( XTREME106_CMDS
TRUCTURE *XtrStructure );

extern int    PostEmptiveInitInit(void);
extern struct Shir *Sursa;
extern char   extractionDestination[_MAX_PATH];
extern unsigned char metoda, COMPRESS_RECOVERY_RECORD;
extern char   Destination[_MAX_PATH];
extern char   Inghetz, Testez, Sterg, Extrag, TestedOk, Listez, Act
ualizez, Repack, Protejez,
              Repar,   ZipCloaking,   Mut,    Incui;
extern char   TableIndex, Solida, AssumeYes, Recursiv;


XTREME106_CMDSTRUCTURE  MyXtrStructure;
#define DLLInitError(a,b,c,d)  if(XtrStructure->ErrorMessageDisplay
Function)  XtrStructure->ErrorMessageDisplayFunction(a,b,c,d);


void FreeOtherStuff(void)
{
  free((struct Shir *)Sursa);
  Fclose(F1, 0, 0);              // everything must be shut off whe
n exiting
  Fclose(F2, 0, 0);              // bugs disappear ..
  F1 = NULL;
  F2 = NULL;
```

InitDLL

```
}

int Init_Xtreme106_Engine ( XTREME106_CMDSTRUCTURE *XtrStructure )
{
  Mut          = 0;
  Incui        = 0;
  Sterg        = 0;
  Listez       = 0;
  Repar        = 0;
  Repack       = 0;
  Testez       = 0;
  Extrag       = 0;
  Inghetz      = 0;
  TestedOk     = 1;
  Protejez     = 0;
  Actualizez   = 0;
  ZipCloaking = 0;

  memcpy((unsigned char *)&MyXtrStructure,
         (unsigned char *)&(*XtrStructure),
         sizeof(XTREME106_CMDSTRUCTURE)); // why should we forget w
onderful thing ?

  memset(Destination,           0, _MAX_PATH);
  memset(extractionDestination, 0, _MAX_PATH);

  switch(XtrStructure->function)
  {
    case  XTREME_ADD         : Inghetz  = 1; break;
    case  XTREME_TEST        : Testez   = 1; break;
    case  XTREME_MOVE        : Mut      = 1;
                               Inghetz  = 1; break;
    case  XTREME_LIST        : Listez   = 1; break;
    case  XTREME_LOCK        : Incui    = 1; break;
    case  XTREME_DELETE      : Sterg    = 1; break;
    case  XTREME_REPAIR      : Repar    = 1; break;
    case  XTREME_EXTRACT     : Extrag   = 1; break;
    case  XTREME_PROTECT     : Protejez = 1;
                               metoda = XtrStructure->protection_met
hod;
                               COMPRESS_RECOVERY_RECORD = XtrStructu
re->compressn_ecc;     // 0 .. 1
                               break;

    case  XTREME_UNPROTECT   : Protejez = 2; break;
```

InitDLL

```
    default :  // there must be an error, undefined function called

/*     DLLInitError(" Wrong number in Xtreme function init  ! ", "\n
", 12,
                  " None of these functions were supplied to me ...
", "\n", 12,
                  " XTREME_ADD,      XTREME_TEST,    XTREME_MOVE,    XT
REME_LIST,    ", "\n", 12,
                  " XTREME_LOCK,     XTREME_DELETE, XTREME_REPAIR, XT
REME_EXTRACT,", "\n", 12,
                  " XTREME_PROTECT, XTREME_UNPROTECT", "\n" , 12
                  " Initialization error ... ", " ", 2, " ", " ", 2)
; // so ...
                  */
    return 1;
    break;
 }

  // check the options !

  Recursiv     = XtrStructure->recursive;
  TableIndex   = XtrStructure->tablesize;
  Solida       = XtrStructure->solid;
  ZipCloaking = XtrStructure->zipCloaking;

  if(Recursiv != 0 && Recursiv != 1)     return 2;
  if(Solida    != 0 && Solida    != 1)   return 3;
  if(TableIndex > 8)                     return 4;
  if(TableIndex == 0) TableIndex++;
  if(Protejez == 1 && (metoda > 3 || (COMPRESS_RECOVERY_RECORD != 0
 && COMPRESS_RECOVERY_RECORD != 1)))    return 5;

  if((Sursa = (struct Shir *)malloc((XtrStructure->MaxDirectories)*
sizeof(struct Shir))) == NULL)  return 7; // not enough memory!
  // if MaxDirectories field is specified, then take care, cause yo
u have to suplly them, or else ....
  for(unsigned int i=0; i < XtrStructure->MaxDirectories; i++)
  {
      memset(Sursa[i].Name,                          0,        _MAX_PAT
H);
          memcpy(Sursa[i].Name, XtrStructure->Sursele[i].Name, _MAX
_PATH);
  }

  //if(XtrStructure->extractionDirectory[0])
  memcpy(extractionDestination, (char *)(XtrStructure->extractionDi
```

InitDLL

```
rectory), _MAX_PATH);
  //if(XtrStructure->Destination[0])
  memcpy(Destination, (char *)(XtrStructure->Destination), _MAX_PAT
H);

  if(Sursa[0].Name[0] == 0 && Destination[0] == 0) return 6;

  if(Extrag == 1 || Sterg == 1 /* --> is somewhere else, don't worr
y || Actualizez == 1  I think there is room for Adaug and test */)
        FileSelection();

  dll_main(); // and execute, too !

  return 0;
}
```

```
#define  BITS                26
#define  MAXCHAR_CONTEXT     35    // MAXIMMUM CONTEXT CHAR NUMBER
#define  DOI_LA(n)         (1<<n)
#define  HALF              DOI_LA(BITS - 2)
#define  FULL              DOI_LA(BITS - 1)

struct contextModel
{
  unsigned int SymbolsNo;
  unsigned int TotalFrequency;
  unsigned int Step;
  unsigned int MaximumFreq;
  unsigned int Freq[MAXCHAR_CONTEXT + 1];
};

struct Fill_In_Data
{
   unsigned int flag;
   unsigned int retValue;
   unsigned int thresholdValue; // please do not change the order o
k
   unsigned int symbolsNumber;
   unsigned int step;
};

void ContextualEncodeBuffer(unsigned int read, unsigned char *Inbuf
, unsigned char *Outbuf, unsigned int  *out);
void ContextualDecodeBuffer(unsigned int read, unsigned char *Inbuf
, unsigned char *Outbuf, unsigned int *OutNo);
```

crc

```
#define INIT_CRC_VALUE  0xffffffffL

extern unsigned int crc_32_tab[];
void updcrc(unsigned char *s, unsigned int n, unsigned int *whatcrc
);
void local_short_crc(unsigned char *s, unsigned int n, unsigned sho
rt *whatcrc); // 2 byte CRC derived from 4 byte CRC
void local_uint_crc(unsigned char *s, unsigned int n, unsigned int
*whatcrc); // 4 byte CRC
```

```
typedef unsigned char  uch;
typedef unsigned       ush;
typedef unsigned int   ulg;

extern unsigned char       *Fereastra;                          // Sl
iding window and suffix table (unlzw)
extern struct NewInteger   *prev;                               // pr
efix for even codes
extern struct NewInteger   *head;                               // pr
efix for odd  codes

//extern unsigned int *prev;                                  // prefix
for even codes
//extern unsigned int *head;                                  // prefix
for odd  codes


extern unsigned int        HASH_BITS;
extern unsigned int        HASH_SIZE;
extern unsigned int        HASH_MASK;
extern unsigned int        WMASK;
extern unsigned int        window_size, string_starting;          /
/ start of string to insert
extern unsigned int        MAX_DIST;
extern unsigned int        H_SHIFT;

extern unsigned int        MIN_MATCH;      //  4,5
extern unsigned int        MAX_MATCH;      //  258
      //258  // The minimum and maximum match lengths
extern unsigned int        MIN_LOOKAHEAD;

int inflate();
int deflate();
void initialize_engine(short);
```

Defuri

```
#include <conio.h>

#define HELPCOL1                    16*8 + 11
#define HELPCOL2                    16*8 + 14
#define HELPTEXT1                   16*8 + 14
#define HELPTEXT2                   16*8 + 11
#define HELPTEXT3                   16*8 + 15

#define VOLUME                      16*8 + 10
#define UNLINKING                   16*8 + 11

#define REGI                        16*8  + 11

#define MENU_COL1                   16*7  + 11
#define MENU_COL2                   16*7  + 0
#define MENU_SELECTED               16*9  + 14
#define MENU_DISABLED               16*7  + 15

#define COMMENT                     16*8 + 11
#define COMMENT_1                   16*8 + 11
#define COMMENT_2                   16*8 + 14
#define COMMENTARIU                 16*8 + 15

#define XTREME_VERSION                      10
#define LOG_FILE_NAME             "klm_.log"

#define CHECKBOXCHAR                     'X'
#define CBOX_STR_COLOR              16*7 + 15
#define CBOXCOLOR                   16*7 + 14
#define CBOX_COLOR                  16*7 + 11

#define RETRY_COD                         1
#define ABORT_COD                         2
#define IGNORE_COD                        0

#define STORED                         0x00
#define STATIC                         0x01
#define DYNAMIC                        0x02

#define ADDING                         0x07
#define TESTING                        0x08
#define EXTRACT                        0x09

#ifdef      __cplusplus
#define     __CPPARGS ...
#else       #define __CPPARGS
```

Defuri

```
#endif


#define ALT_MENU                    16*1  + 14
#define ALT_COL1                    16*1  + 11
#define ALT_COL2                    16*1  + 10
#define ARC_ALT_MENU                16*1  +  2

#define ADD_COL1                    16*8 + 15
#define ADD_COL2                    16*8 + 14

#define SHORTCUT                    16*8 + 11
#define PAROL_COL                   16*8 + 14
#define RAM                         16*8 + 15
#define COPYR                       16*8 + 14
#define OSPATH                      16*8 + 14
#define FREESPACE                   16*8 + 15

#define EDIT                        16*8 + 15          // Dialo
g : Volatile text
#define WRITE                       16*8 + 14          //  Color
s : UnVolatile

#define PERCENT                     16*8 + 2
#define ADDPERCENT                  16*8 + 14
#define TSTPERCENT                  16*8 + 2

#define ERROR_COL1                16*RED + 15
#define ERROR_COL2                16*RED + 14

#define PASSW_COL1                16*RED + 15
#define PASSW_COL2                16*RED + 14

#define INSERT_FOND                 16*8 +  2
#define INSERT_COL1                 16*7 + 15
#define INSERT_COL2                 16*7 + 14
#define INSERT_CHAR                 16*8 + 14

#define INSERT                         0x52
#define HOME                           0x47
#define DEL                            0x53
#define END                            0x4f
#define TAB                            0x09
#define ST                               75
#define DR                             0x4d
#define UP                             0x48
```

Defuri

```
#define DN                              0x50

#define NOSCROLL                          2
#define DOSCROLL                          3
#define NOINFO                            0
#define DOINFO                            3

#define CLOCK                             1
#define XCLOCK                           73
#define YCLOCK                            1
#define CLOCK_COLOR                 16*8 + 11

#define DOWN                             20
#define ATTRIB_CHAR                     178    // 0xfb
#define MAXSCROLL_LEN                    17
#define INSERT_BUTTON                     2

#define ATTRIB_CHAR                     178      // 0xfb
#define MAXSCROLL_LEN                    17
#define DOWN                             20

#define JOS                               1
#define SUS                               2
#define MAX_MOUSE_SENSITIVE_TEXT         50

#define ATTRIBCHAR                      254
#define NoTiceTime                       30   //Seconds

#define DRIVE_NORMAL                16*8  + 11

#define ADDCOL                      16*7  + 14
#define ADDFOND                     16*7  + 15

#define ARCHIVE                     16*8  + 11

#define FOND                        16*8  + 15
#define INFO                        16*8  + 14
#define TEXTINFO                    16*8  + 14   // Name, Size,
..

#define SCROLL_ARROW                    254
#define SCROLL_FOND_CH                  219
#define SCROLL_KNOT_COL             16*8 + 13
#define SCROLL_FOND_COL             16*8 +  7
#define SCROLL_FOND_SELECTED              0
#define SCROLL_ARROW_COL            16*8 + 15
```

Defuri

```
#define FUNCTIONCOL                     16*3 +  4
#define STATUSBARCOLOR                  16*3 +  8

#define FILE_NORMAL                     16*8 + 15   // Atribut Fisie
r normal
#define FILE_INSERTED                   16*8 + 14   // Atribut Fisie
r inserat
#define FILE_NORMAL_SELECTED            16*9 + 14   // Atribut Fisie
r normal selectat
#define FILE_INSERT_SELECTED            16*9 +  3   // Atribut Fisie
r inserat selectat

#define DIRECTORY_NORMAL                16*8 + 11
#define DIRECTORY_INSERTED              16*8 + 14
#define DIRECTORY_NORMAL_SELECTED       16*9 + 14
#define DIRECTORY_INSERT_SELECTED       16*9 +  3

#define SENSITIVE_TEXT_NORMAL           16*7 +  0
#define SENSITIVE_FUNCTIONS_NORMAL      16*3 +  0
#define SENSITIVE_TEXT_SELECTED         16*3 + 14

#define TOATE                           FA_DIREC+FA_RDONLY+FA_HIDDEN
+FA_SYSTEM+FA_ARCH

#define vseg 0xb800

#define Year(ts)          ((unsigned int) ((ts >>  9) & 0x7f) + 80)
//+1980
#define Month(ts)         ((unsigned int) (ts >>  5) & 0x0f)
/* 1..12 means Jan..Dec */
#define Day(ts)           ((unsigned int) (ts >>  0) & 0x1f)
/* 1..31 means 1st..31st */
#define Hour(ts)          ((unsigned int) (ts >> 11) & 0x1f)
#define Min(ts)           ((unsigned int) (ts >> 5) & 0x3f)
#define Sec(ts)           ((unsigned int) ((ts & 0x1f) * 2))

extern   void ASTEAPTA(void);
```

Exports

```
#define XTREME_ADD                    1
#define XTREME_TEST                   2
#define XTREME_MOVE                   3
#define XTREME_LIST                   4
#define XTREME_LOCK                   5
#define XTREME_DELETE                 6
#define XTREME_REPAIR                 7
#define XTREME_EXTRACT                8
#define XTREME_PROTECT                9
#define XTREME_UNPROTECT             10

struct FileName
{
   char *Name;                        // directory to add ...
   char  File;                        // file or directory
   char  Path1Or2;                    // since a path is required i
n an interface, here is its index
   char  NotUsed1; // 4
   char  NotUsed2; // 1
};


typedef struct
{
  char solid;
  char function;                      // ALL those above !
  char tablesize;
  char recursive;
  char zipCloaking;

  char sfx;
  char compressn_ecc;                 // 0 .. 1
  char protection_method;             // 0 .. 3

  char *Path1;                        // files path 1 ...
  char *Path2;                        // files path 2 ...
  struct FileName Sursele[10000];

  char *Destination;                  // archive name ...
  char *extractionDirectory;          // extract to directory ...
  unsigned int MaxDirectories;


  void (*EndDLLJob)                       (void);
  int  (*Utility_Function)                (char *mess,       int
```

Exports

```
 but_no);

 void (*ErrorMessageDisplayFunction)        (char *message1,   char
*delimit1, int color1,
                                             char *message2,   char
*delimit2, int color2,
                                             char *message3,   char
*delimit3, int color3,
                                             char *message4,   char
*delimit4, int color4);

 void (*MinorStatusMessageDisplayFunction) (char *status1,    char
*delimit1, int color1);//,
                                             //char *status2,    ch
ar *delimit2, int color2);

 void (*MajorStatusMessageDisplayFunction) (char *status1,    char
*delimit1, int color1,
                                             char *status2,    char
*delimit2, int color2,
                char *status3,    char *delimit3, int color3,
                char *status4,    char *delimit4, int color4,
                                            int   procent);
} XTREME106_CMDSTRUCTURE;

extern XTREME106_CMDSTRUCTURE  MyXtrStructure;
```

huffman

```
#define MAX_CHARACTERS      256
#define ROOTPLUSONE         511
#define ROOT                510
#define MAX_FREQ          0x8000
#define MAX_READ         0x32000


struct  HuffmanTree
{
    int Spl_Freq[512];
    int Spl_Dad [512 + 256];
    int Spl_Son [512];
};

extern unsigned int  outcount;
extern unsigned char OutBuf[MAX_READ*2]; // prevent expanding !


//
//  EncodeBuffer(Size, Undefined,  Undefined2, &X);
//  DecodeBuffer(Size, Undefined, Undefined2, &Xk); <--- Size has t
he same value here
//  Ex -> Encode(66000,...) --->  4000                    just like
at the encoding
//        Decode(66000,...) ---> 66000

void EncodeBuffer(unsigned int Size, unsigned char *Buffer, unsigne
d char *OutBuff, unsigned int *outcnt);
void DecodeBuffer(unsigned int Size, unsigned char *Buffer, unsigne
d char *OutBuff, unsigned int *outcnt);
```

```
/* Common procedures and functions for both Encode and Decode */

struct   Tree {
                short    Spl_St [256];
                short    Spl_Dr [256];
                short    Spl_Dad[512];
               };

void InitModel(void);
void De_Init_Buf(void);
void MFillName(unsigned char Chr);
void InitializeMarkovStates(void);
void SemiSplay(unsigned char Ch);
void BreakChars(unsigned char Ch);
void MarkovEncode(unsigned int Size, unsigned char *Buffer, unsigne
d char *OutBuff, unsigned int *outcnt);
void MarkovDecode(unsigned int Size, unsigned char *Buffer, unsigne
d char *OutBuff, unsigned int *outcnt);
```

171

```
extern unsigned long header_pos; // the main variable for
                                 // non-solid archives communicatio
n !

int  TestFrostedFiles(void);
int  DeleteNonSolidArchive(char *files);
int  UpdateNonSolidArchive(char *files, char _to_what_type);


int  Write_Fake_Header(void);
int  Write_Real_Header(FILE*);

void MeltFile(void);
void NonSolidListFiles(char *files);
void Freeze(char *FileMask, char repack);
void PackFile(char *name, unsigned int size);
```

Recovery

```
#define NON_CONSECUTIVE_NO                  11 // 3, 5, 7
#define PROTECTED_BUFFER_NO                 21 // the bigest values
// 5, 7, 11
#define PROTECTED_BUFFER_SIZE              512

struct DataRecovery
{
  unsigned short  CRC;
  unsigned char   buffer   [PROTECTED_BUFFER_SIZE];
};

struct Protect
{
   unsigned short      n_subsectors;
   unsigned int        n_buffer_size;
   struct DataRecovery *recovery_       ;// [NON_CONSECUTIVE_NO];
   unsigned short      *SubSectorCrc    ;// [NON_CONSECUTIVE_NO * PR
OTECTED_BUFFER_NO];
};

struct Data
{
  unsigned short  CRC;
  unsigned char   buffer [PROTECTED_BUFFER_SIZE];
};

struct ReadStructure
{
   struct Data Date[PROTECTED_BUFFER_NO];
};

extern unsigned char metoda, COMPRESS_RECOVERY_RECORD;

int CheckDataRecovery(FILE *FF);
int WipeCorrectionTables(FILE *FF);
int ReadProtectStructure(FILE *FF, char read);
int WriteCorrectionTable(FILE *FF, char flush);
int WriteProtectStructure(FILE *FF, char flush);
int ReadCorrectionTable(FILE *FF, unsigned int FSize);
int DamageDetected(unsigned char *Table, unsigned int SectorI, unsi
gned int Size, unsigned short *errorIndex);

void Repair();
void ComputeDataRecoveryRecord(FILE *FF);
```

Recovery

```
void InitializeProtection(unsigned char met, unsigned int size, cha
r protect);
void Repair(unsigned char *Table, unsigned int bad_subsector, unsig
ned int TableSize);
void InitializeCorrectionForFile(FILE *FF, unsigned int size, char
copy, unsigned char metode);
```

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by tester.rc
//
#define IDD_TESTER_DIALOG               102
#define IDR_MAINFRAME                   128
#define IDC_STATIC002                   1000
#define IDC_STATIC001                   1001
#define IDC_STATIC003                   1002
#define IDC_STATIC_004                  1003
#define IDC_STATIC_005                  1004
#define IDC_STATIC_100                  1005
#define IDC_STATIC_101                  1006
#define IDC_STATIC_102                  1007
#define IDC_STATIC_103                  1008
#define IDC_LIST1                       1009
#define IDC_LIST2                       1010

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        129
#define _APS_NEXT_COMMAND_VALUE         32771
#define _APS_NEXT_CONTROL_VALUE         1011
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

Solid

```
struct Solid_XTREME_Header
 {
   unsigned long  _Time_Date;            // 8
   unsigned int   _SizeUnCom;            // 4
   unsigned char  _NamLen;               // 1
   unsigned char  _Atrib;                // 1
   unsigned char  _NOT_USED1;            // 1
   unsigned char  _NOT_USED2;            // 1
   unsigned int   _SolidCRC;             // 2
 };

//  Since opening a file does not mean actually that we are going t
o
// preserve its full name in the archive, then the easiest solution
// will be to keep two names for the file, which is actually the si
ngle
// solution!
//  It is the best implementation for cases related the interfaces
// where we are in a directory and we are adding sone files ....
// there are not supposesd to be added with FullPath!

struct SolidFileType
{
   unsigned long        time_date;
   unsigned int              size;
   char                     atrib;
   char                  selected;

   char    ArchiveKeptName[_MAX_PATH];
   char           RealName[_MAX_PATH];
   char               ext[_MAX_EXT];
};

int   Write_Solid_CRC(void);
int   Write_Solid_Header(void);
int   Read_Solid_Header(void);

extern unsigned long TotalSolidSize;
extern unsigned int  SolidN, Solid_CRC;
extern struct SolidFileType *SolidArray;

int   SolidInit();
int   TestSolidArchive(void);
int   SolidDeleteFiles(char *);
```

Solid

```
int   UpdateSolidArchive(char *files, char _to_solid_or_normal);

int   SolidMelt(void);
void  SolidExtract(void);
void  SolidFreeze(char *);
int   SolidListFiles(char *files);
void  UpdateSolidArray(unsigned int SolidKa);
void  SolidFillTheName(unsigned int, char *, unsigned int, unsigned
 long, unsigned char);
void  SolidRePack(unsigned int no, char *fromWhere, char update);
```

```
#include <stdio.h>

#define LITBUF          56*1024
#define LITBUF_TO_ALLOC 64*1024 // prevent expanding


/*
  Okay, please beware that these values are checked in case of test
ing
  an archive 'stricata'! So, within 1K, the error must appear to be
 detectable
  So, remember that when you change these ones!
 */

extern  FILE  *F1, *F2;

struct Huff_Block
 {
    unsigned char  _RestBits        :  5;
    unsigned char  _Block_Type_0    :  1;   // Bits
    unsigned char  _Block_Type_1    :  1;   // Literals
    unsigned char  _Block_Type_2    :  1;   // Lenghts
    unsigned char  _Block_Type_3    :  1;   // Distances
    unsigned char  _Block_Type_4    :  1;   // RestBuf2
    unsigned int   _NOT_USED        :  6;
    unsigned int   _CRC_total;              //  One for all
 };

void Tree_Start();
void Tree_Flush();
int  Read_Block();
void bits_flush(void);
int  Tree_Encode(int, int);
void send_bits(unsigned short value, short length);
void get_bits(unsigned short *value, short length, bool ok);

extern unsigned char restbits;

extern unsigned char   *BitBuf;   extern unsigned int BitBufC;
extern unsigned char   *LenBuf;   extern unsigned int LenBufC;
extern unsigned char   *LitBuf;   extern unsigned int LitBufC;
extern unsigned char   *DistBuf;  extern unsigned int DistBufC;
extern unsigned char   *RestBuf2; extern unsigned int RestBufC2;  /
/ the compressible rest
extern unsigned short  *RestBuf1; extern unsigned int RestBufC1;
```

```
#include <io.h>
#include <stdio.h>
#include <stdlib.h>

extern short ratio;
extern char  ZipCloaking;

#ifndef  CheckSl
 #define  CheckSlash(t)  if(t[strlen(t) - 1] != '\\' && t[0])  strc
at(t, "\\");
 #define  CheckSl
#endif

struct TableData
{
   unsigned short WSIZE_INDEX;          //  Where WSIZE is WSIZE_I
NDEX * 0x8000
   unsigned short MIN_MATCH;            //  The too far !
   unsigned short MAX_MATCH;            //  The longest Match leng
th
   unsigned short HASH_BITS;            //  The bits for hashing !
   unsigned short TOO_DISTANT;          //  If it's too distant ab
ort !
   unsigned short D_REST_BITS;
   char Descriptor[50];
};

struct NewInteger
{
   unsigned short Li;                   //  the  0x....AABB
   unsigned char  Hc;                   //  the  0x..CC....
};

struct HashNode
 {
   unsigned int   _Info;                //  4
   unsigned int   _NextLink;            //
};

// this structure was cloned in 'export.h'
struct ARH_Header // if you modify this, please update the export.h
 file, please
 {
   char           _Name[8];             // 8 = "ULEB106\0"
   unsigned int   _SolidN;              // very good
   unsigned char  _Version;             //
```

```
   unsigned char  _PassWord;              //
   unsigned char  _Commented;             //
   unsigned char  _Locked;                //
   unsigned char  _Vol;                   //
   unsigned char  _HeaderCRC;             //
   unsigned int   _recovery_Filesize;     // required by damage prot
ection
   unsigned int   _recovery_Method;
   unsigned int   _recovery_Compressed;

   unsigned char  _Not_Used         : 2;
   unsigned char  _Solida            : 2;
   unsigned char  _SolidaTableSize : 4;  //  One for all
   unsigned char  _NOT_USED1        : 8;

   unsigned int   _UHeaderCRC;            //  real 32bit CRC for th
e header reconessence!

 };

struct XTREME_Header
 {
   short _Name       : 3;   // 101
   short _Flag       : 3;
   short _Directory  : 2;
   short _Table      : 5;
   short _NotUsed    : 3;

   unsigned char  _NamLen;                // 1
   unsigned char  _HeaderCRC;             // 1

   unsigned short _Time;                  // 2
   unsigned short _Date;                  // 2
   unsigned int   _SizeUnCom;             // 4
   unsigned int   _NextFile;              // 4
   unsigned int   _FileCRC;               // 4
 };

struct Shir
{
   char Name[_MAX_PATH];
};

#define PutC(Cod)  {TotalWrit ++; putc(Cod, F2); }

#define SprintfBigNumber(sir,no)  { if(no  <=   999) sprintf(sir,
```

```
      "%03d",                                    no%1000); \
                                                          else if(no <
= 999999) sprintf(sir,    "%03d,%03d",                (no/1000)%1000,
 no%1000); \
                                                            else
             sprintf(sir, "%d,%03d,%03d",  no/1000000, (no/1000)%1
000, no%1000); }

extern unsigned char *Fereastra;
extern unsigned int   TOO_DISTANT;
extern unsigned int   WSIZE;
extern unsigned short D_REST_BITS;
extern unsigned int   XXXInBufC, XXXOutBufC;
extern unsigned char *XXXInBuf, *XXXOutBuf;
extern struct ARH_Header ARH_H;
extern struct XTREME_Header XTREME_H;

int  Read_ZIP_ARH_Files();
void Read_ZIP_ARH_header();


int  FileSelection(void);

void TryToRepair();
void GetCurDir(char *temp);
void Register_File_List(char *FileMask);
void GetTemporaryDirectory(char *buffer);
void Split(char *Name, char *drv, char *dir, char *fnam, char *ext)
;
void ComplexSplit(char *Name, char *drv, char *dir, char *fnam, cha
r *ext);


int  Verify(char *, char *, char, char);
int  file_exists(char *disk, char *filename);
int _unlink_path(char *path, char kill_directory);

int  Check_ARH_Header(struct ARH_Header *Ar);
int  FWrite_ARH_Header(struct ARH_Header *Ar, FILE *F);
int  CreateTemporaryArchive(char *DestinationDir, char *temp_name);

extern void GetArchiveType(char *archive_type);
extern void window_fill_in(void);
extern void Error(short errorindex, char *additional_text);
extern struct Shir *Sursa;
extern char Destination[_MAX_PATH];
```

```
extern struct  ARH_Header ARH_H;

extern void Fclose(FILE *, unsigned int, unsigned int);
extern void PackFile(char *, unsigned int);
extern int  OpenFile(char *, unsigned int, unsigned char);

int Getc();
int flush_outbuf(void);
int CheckStruct(char *P);

int Write_Archive_Header(void);
int write_buf(char *buf, unsigned cnt);
int Read_Archive_Header(char initialize_tables);
int read_from_file(unsigned char *buf, unsigned int size);
short IntoarceProcent(unsigned long, unsigned long);
unsigned int fread_crc(unsigned char *,   unsigned int);

char *short_str(char *, int);

void Print_Start();
void RepairArchive(void);
void ReadjustTables(void);
void SkipToNextFile(FILE *);
void TestRecoveryRecord(void);
void DamageProtectorArchive(void);
void flush_window(unsigned int Cati);
void InitFilesNameBuffer(unsigned char*, unsigned int);


extern bool write_melt;
extern char InFile[_MAX_PATH];
extern char tmp[_MAX_PATH], tmp2[_MAX_PATH];

extern unsigned int afis_start;        /* contor to count */
extern unsigned int count_start;       /* contor to print */
extern unsigned int SolidK, SolidN;
extern long TotalRead, TotalWrit, TotalSum;

extern FILE  *F1, *F2;
extern char   TableIndex, Solida, AssumeYes, Recursiv, TestedOk;

extern unsigned int prev_length;
extern unsigned int string_starting;
extern unsigned int afis_start;          /* contor to count */
extern unsigned int count_start;         /* contor to print */
extern unsigned int match_start;
```

```
extern unsigned int in_advance_reading;
extern unsigned short max_chain_length;
extern unsigned short max_lazy_match;
extern unsigned short good_match;
extern int nice_match, EndOfFile;
extern unsigned int hash_header;  // hash index of string to be ins
erted
extern unsigned int MaxNodes, MAXNODES;

extern char Inghetz, Testez, Sterg, Extrag, Repack, TestedOk, Liste
z, Actualizez, Mut;

#define memory_request_alloc(block_type, the_array, size_of_request
)  the_array = (block_type*)calloc((size_t)(((size_of_request)+1L)/
2), 2*sizeof(block_type));
#define memory_request_free(the_array) {if (the_array != NULL) free
(the_array), the_array=NULL;}

extern unsigned int  TOO_DISTANT, WSIZE, window_size, HASH_BITS, HA
SH_SIZE, HASH_MASK, WMASK, MAX_DIST, H_SHIFT, MIN_MATCH, MAX_MATCH,
 L_REST_BITS, MIN_LOOKAHEAD;
extern unsigned char HASH_MAX;
extern char extractionDestination[_MAX_PATH];
```

```
//ZIPHEADS


struct ZIP_Header
{
  unsigned int   signature;          //  4 local file header signat
ure     4 bytes  (0x04034b50)
  unsigned short version;            //  6 version needed to extrac
t       2 bytes
  unsigned short general_bit;        //  8 general purpose bit flag
        2 bytes
  unsigned short compression_method; // 10 compression method
        2 bytes
  unsigned short time_;              // 12 last mod file time
        2 bytes
  unsigned short date_;              // 14 last mod file date
        2 bytes
  unsigned int   crc32;              // 18 crc-32
        4 bytes
  unsigned int   compressed;         // 22 compressed size
        4 bytes
  unsigned int   uncompressed;       // 26 uncompressed size
        4 bytes
  unsigned short filename1;          // 28 filename length
        2 bytes
  unsigned short extra_len;          // 30 extra field length
        2 bytes
};

struct ZIP_Central_Directory_Header
{
  unsigned int   signature;        // central file header signature
   4 bytes  (0x02014b50)
  unsigned short ver;              // version made by
2 bytes
  unsigned short ver_extr;         // version needed to extract
2 bytes
  unsigned short general_bit;      // general purpose bit flag
2 bytes
  unsigned short compress_m;       // compression method
2 bytes
  unsigned int   time_date;        // last mod file time
2 bytes unsigned short   last mod file date              2 bytes
  unsigned int   crc32;            // crc-32
4 bytes
  unsigned int   compressed;       // compressed size
```

```
4 bytes
  unsigned int   uncompressed;  // uncompressed size
4 bytes
  unsigned short file_n;        // filename length
2 bytes
  unsigned short extra_field;   // extra field length
2 bytes
  unsigned short file_comment;  // file comment length
2 bytes
  unsigned short disk_no;       // disk number start
2 bytes
  unsigned short internal_attr; // internal file attributes
2 bytes
  unsigned int   external_attr; // external file attributes
4 bytes
  unsigned int   relat_offset;  // relative offset of local header
4 bytes
};

struct ZIP_End_Of_Central_Directory_Header
{

  unsigned int   signat_end;          //  end of central dir signatu
re    4 bytes  (0x06054b50)
  unsigned short this_id;             //  number of this disk
      2 bytes
  unsigned short all_no;              //  number of the disk with th
e     start of the central directory  2 bytes
  unsigned short total_entries_here; //  total number of entries in
      the central dir on this disk    2 bytes
  unsigned short total_entries;       //  total number of entries in
      the central dir                  2 bytes
  unsigned int   size_of_cdh;         //  size of the central direct
ory   4 bytes
  unsigned int   start_cdh;           //  offset of start of central
      directory with respect to        the starting disk number
   4 bytes
  unsigned short comment_size;        //  zipfile comment length
      2 bytes
                                      //  zipfile comment (variable
size)
};


extern ZIP_Header                              ZIP_H;
extern ZIP_Central_Directory_Header          ZIP_CDH;
```

```
extern ZIP_End_Of_Central_Directory_Header  ZIP_CDHE;


int WriteFirstZipHeader(void);
int WriteZipHeaders(long *AllWritten);
```

WHAT IS CLAIMED IS:

1. A method of compressing a plurality of files, said method comprising:

examining said plurality of files to determine data characteristics that correspond to said plurality of files;

determining ranking orders for said plurality of files according to said data characteristics;

combining said plurality of files into a unified file at least according to said ranking orders; and

compressing said unified file.

2. The method of Claim 1, wherein compressing said unified file comprises compressing said unified file using a dictionary method.

3. The method of Claim 1, wherein compressing said unified file comprises:

compressing said unified file using a dictionary method to produce an output data file; and

compressing said output data file.

4. The method of Claim 1, wherein determining ranking orders comprises listing said plurality of files in a file order list.

5. The method of Claim 1, wherein examining said plurality of files comprises examining file extensions of said plurality of files.

6. The method of Claim 1, wherein examining said plurality of files comprises examining the file names of said plurality of files.

7. The method of Claim 1, wherein examining said plurality of files comprises examining the file sizes of said plurality of files.

8. The method of Claim 1, wherein examining said plurality of files comprises:

determining whether at least one of said plurality of files includes predominantly text;

determining whether at least one of said plurality of files includes predominantly numbers;

determining whether at least one of said plurality of files includes predominantly image data;

determining whether at least one of said plurality of files includes predominantly audio data; and

determining whether at least one of said plurality of files includes predominantly video data.

9. The method of Claim 1, wherein said compressing said unified file comprises:

compressing said unified file to produce a output data file; and

combining said output data file with one or more ZIP support portions to produce a ZIP-recognized output file, said ZIP support portions conforming to a ZIP file format.

10. The method of Claim 2, wherein compressing said unified file further comprises using a dictionary that is larger than at least one of said plurality of files.

11. The method of Claim 2, wherein compressing said unified file further comprises using a dictionary that is larger than each of said plurality of files.

12. The method of Claim 2, wherein compressing said unified file further comprises using a dictionary of a pre-determined dictionary size.

13. The method of Claim 2, wherein compressing said unified file further comprises using a dictionary of a dynamically determined dictionary size.

14. The method of Claim 2, wherein compressing said unified file further comprises using a dictionary that is larger than a dictionary size unit number.

15. The method of Claim 3, wherein compressing said output data file further comprises compressing said output data file using a statistical method.

16. The method of Claim 3, wherein compressing said output data file further comprises:

separating said output data file into multiple sections;

compressing said sections using at least one compression method to produce compressed results for said sections; and

combining said compressed results into a final compressed result.

17. A system for compressing a plurality of files, said system comprising:

an examination module configured to examine said plurality of files to determine data characteristics that correspond to said plurality of files;

an ordering module configured to determine ranking orders for said plurality of files;

a combining module configured to combine said plurality of files as a unified file at least according to the ranking orders of said plurality of files; and

a compressing module configured to compress said unified file using a first compression method.

18. The system of Claim 17, wherein said compressing module is further configured to produce an output data file.

19. The method of Claim 17, wherein said first compression method is a dictionary method.

20. The method of Claim 17, wherein said ordering module is further configured to list said plurality of files in a file order list.

21. The system of Claim 18, wherein said compressing module is further configured to compress said output data file using a second compression method.

22. The system of Claim 18, wherein said first compression method and said second compression method are different.

23. The system of Claim 18, wherein said first compression method and said second compression method are the same.

24. The system of Claim 18, wherein said compressing module is further configured to combine said output data file with one or more ZIP support portions to produce a ZIP-recognized output file, said ZIP support portions conforming to a ZIP file format.

25. The system of Claim 19, wherein said compressing module is further configured to dynamically determine a dictionary size.

26. A system for compressing a plurality of files, said system comprising:

means for examining said plurality of files to determine data characteristics that correspond to said plurality of files;

means for determining ranking orders for said plurality of files;

means for combining said plurality of files as a unified file at least according to the ranking orders of said plurality of files; and

means for compressing said unified file using a first compression method.

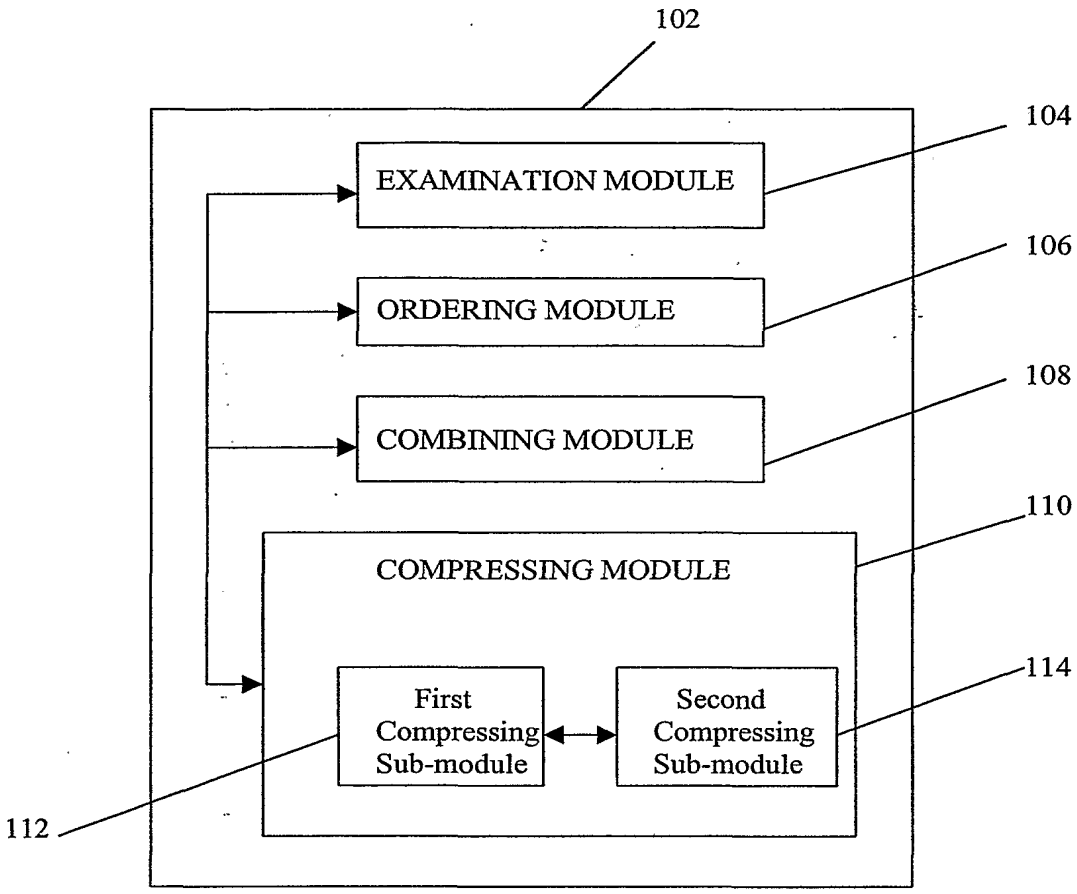

102
104
EXAMINATION MODULE
106
ORDERING MODULE
108
COMBINING MODULE
110
COMPRESSING MODULE
First Compressing Sub-module
Second Compressing Sub-module
114
112
MULTIPLE-FILE COMPRESSION SYSTEM

FIGURE 1

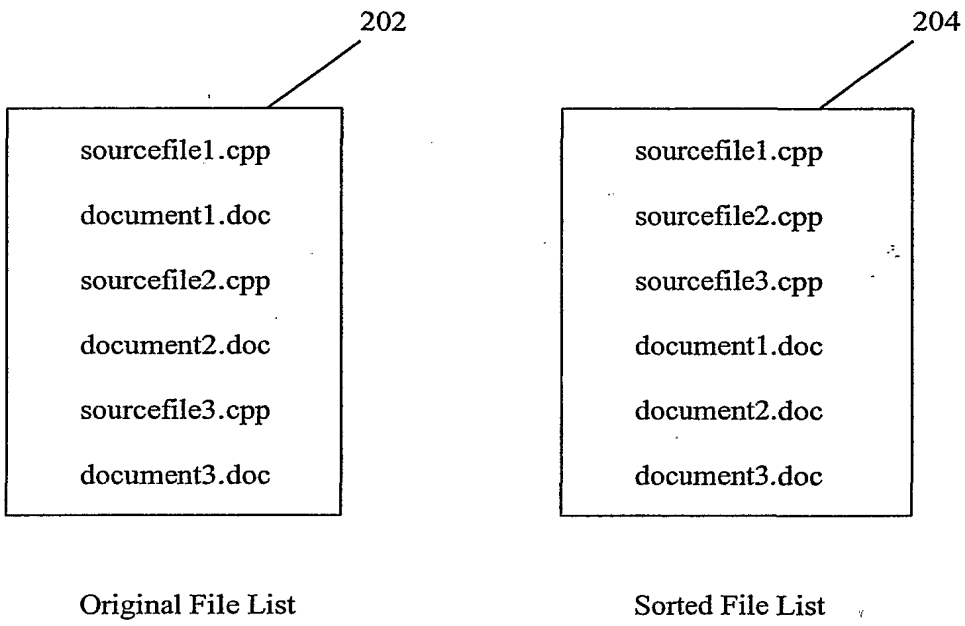

202
sourcefile1.cpp
document1.doc
sourcefile2.cpp
document2.doc
sourcefile3.cpp
document3.doc
Original File List
204
sourcefile1.cpp
sourcefile2.cpp
sourcefile3.cpp
document1.doc
document2.doc
document3.doc
Sorted File List

FIGURE 2

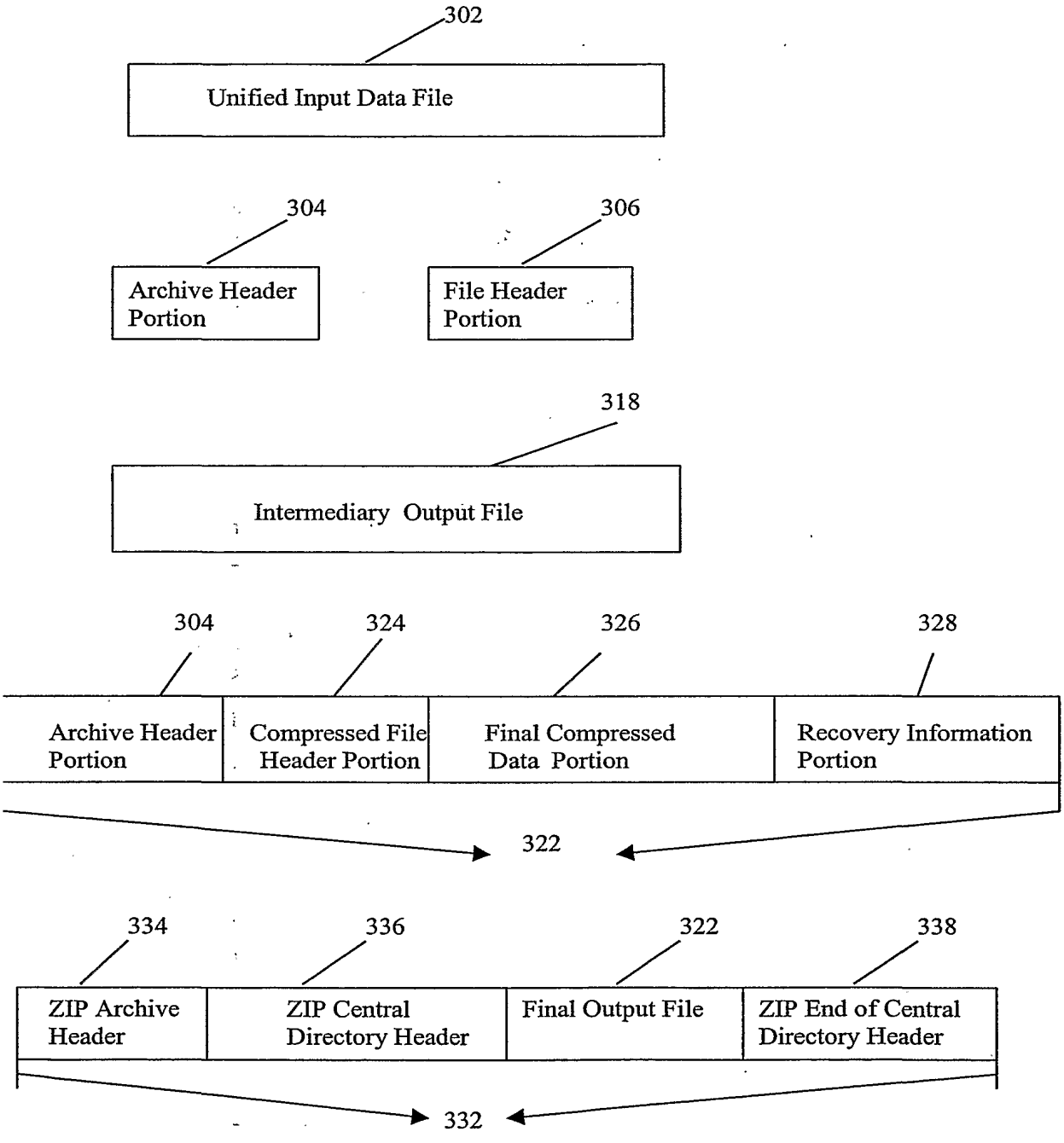

302
Unified Input Data File
304
Archive Header Portion
306
File Header Portion
318
Intermediary Output File
304
324
326
328
Archive Header Portion
Compressed File Header Portion
Final Compressed Data Portion
Recovery Information Portion
322
334
336
322
338
ZIP Archive Header
ZIP Central Directory Header
Final Output File
ZIP End of Central Directory Header
332

FIGURE 3

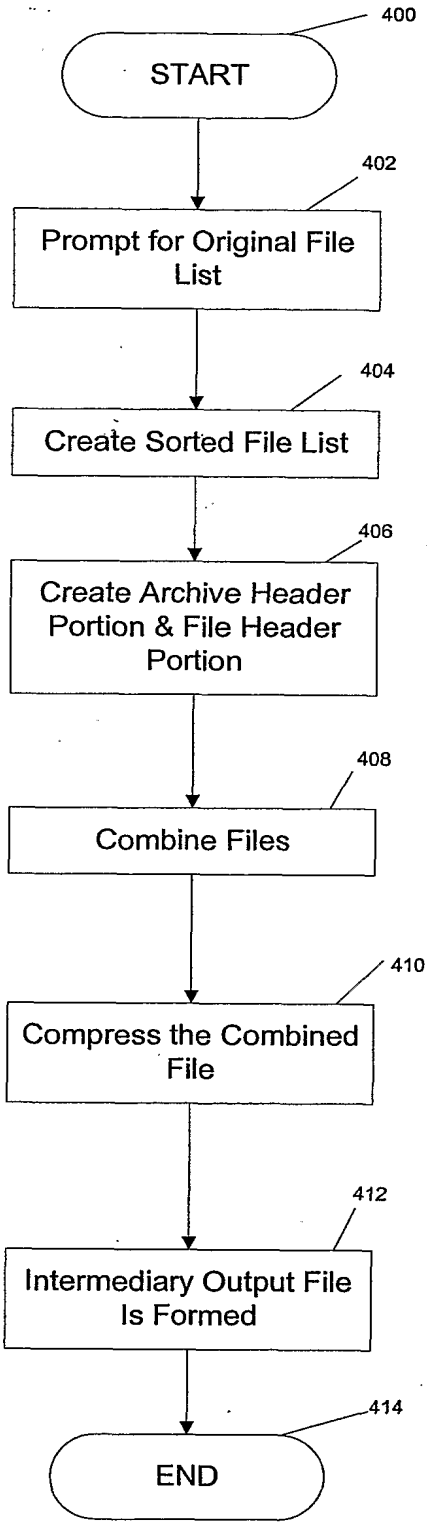

400
START
402
Prompt for Original File List
404
Create Sorted File List
406
Create Archive Header Portion & File Header Portion
408
Combine Files
410
Compress the Combined File
412
Intermediary Output File Is Formed
414
END

FIGURE 4

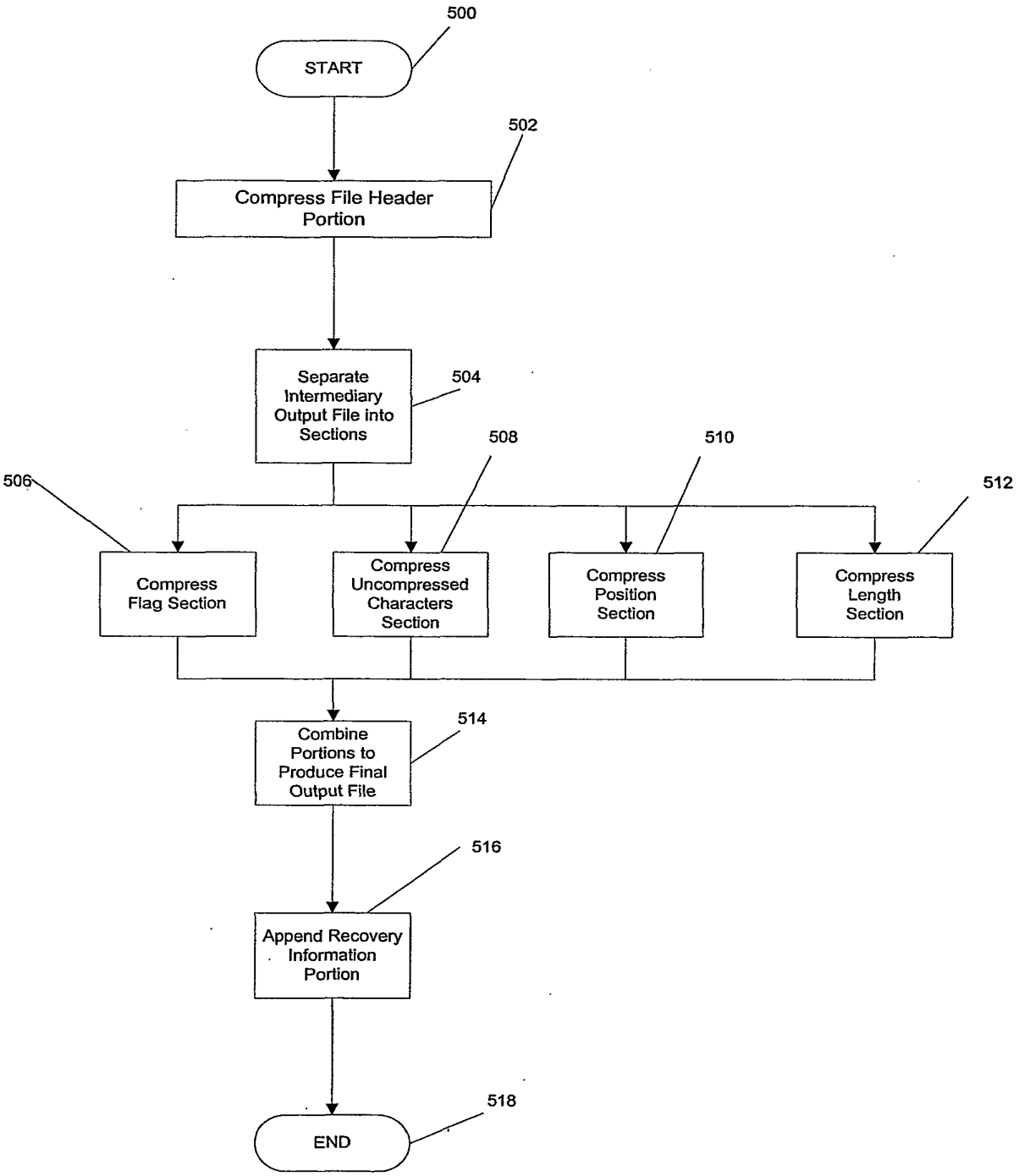

500
START
502
Compress File Header Portion
Separate Intermediary Output File into Sections
504
506
508
510
512
Compress Flag Section
Compress Uncompressed Characters Section
Compress Position Section
Compress Length Section
Combine Portions to Produce Final Output File
514
516
Append Recovery Information Portion
518
END

FIGURE 5

INTERNATIONAL SEARCH REPORT

International A| ion No
PCT/US 01/00424

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 H03M7/30 H03M7/40

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 H03M G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, INSPEC

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	WO 98 39699 A (INTELLIGENT COMPRESSION TECHNO) 11 September 1998 (1998-09-11) page 2, line 12 -page 4, line 5 page 33, line 13 -page 34, line 3	1-26
A	US 5 704 060 A (DEL MONTE MICHAEL G) 30 December 1997 (1997-12-30) column 5, line 5 - line 40; figure 1 column 18, line 19 - line 32; figure 5 column 23, line 20 - line 62	1-26
A	SALOMON D: "DATA COMPRESSION: THE COMPLETE REFERENCE" NEW YORK, NY: SPRINGER,US, 1998, pages 101-162,357-360, XP002150106 ISBN: 0-387-98280-9 paragraphs '3.19!,'3.20!	1-26

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
25 May 2001	01/06/2001

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Georgiou, G

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 01/00424

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	NELSON M R: "JAVA AND THE ZIP FILE FORMAT" DR. DOBB'S JOURNAL,M&T PUBL., REDWOOD CITY, CA,,US, vol. 22, no. 12, December 1997 (1997-12), pages 50,52-54,102, XP000938049 ISSN: 1044-789X paragraph 'The Zip Format! figure 1 -----	1-26

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 01/00424

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9839699 A	11-09-1998	EP 0965171 A	22-12-1999
US 5704060 A	30-12-1997	NONE	